UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

SISTEMA DE INDEXACIÓN DE VB.NET PARA IMPLANTAR PROCESOS DE REUTILIZACIÓN DE UML

Tutor: Juan Llorens Morillo

Autor: Ignacio Herrero Encinar

Octubre, 2010

“You can get it if you really want; but you must try, try and try, try and try. You success at last.

Rome wasn’t built in a day”

Jimmy Cliff, extraído de la canción “You can get it if you really want”

AGRADECIMIENTOS

Ser de bien nacidos es agradecido. Antes de todo indicar que se me van a olvidar personas en este punto pero bueno la vida es así y saben que permanecen en mis más paganas oraciones matutinas y nocturnas.

Como no, los primeros a los que tengo que agradecer y mucho, es a mis padres, José Luis y Mari Carmen, sin los cuales no estaría aquí, y las principales personas por las que siento necesidad de mejorar día a día, y tener la posibilidad de algún día lejano abandonar este mundo sin que nadie me recuerde por lo malo si no por lo bueno. Me servís de ejemplo en cada paso que doy en la vida, a pesar de que a veces crea que no llegaré nunca a vuestro nivel de humildad.

Por supuesto a mi Ana, la única mujer que me ha hecho sentir realmente especial y válido, la que me regala sus ojos azules todas las mañanas, que son mi amanecer.

Por supuesto a mis compañeros de aventuras finlandesas, los que puedo considerar las mejores personas que me llevo de mi largo periplo universitario: Juan y Luis, Luis y Juan. Antes o después tendremos que volver a ver aquel Sol que nunca se ponía o en su defecto a darnos un paseo por el helado mar báltico.

A todas las personas que conocí en Finlandia y que me ayudaron a integrarme en ese ambiente: Los Victores, Erika, Erik y su familia, a José y David mis dos amigos galleos, a Miquel Jordi y a su arroz con costra, a Severino (Sewerin), a todos los componentes del FC Guiri United, a todas las suecas y finlandesas del Arken….

Gracias a todas las amistades nuevas que hice en la universidad: Arturo, Carlos, Adri, David, Samu, Pablo…Me estoy dejando muchísimos pero bueno es lo que tiene el que cada vez que nos reunamos acabemos bastante perjudicados.

Gracias a mis hermanos que me han servido como ejemplo en todo lo que he hecho en la vida, aunque desafortunadamente para mí siempre me hayan puesto el listón muy alto. A pesar de ello siempre sabréis que al menos yo tengo más gracia, y soy más joven.

A todos los profesores y bedeles de la Universidad Carlos III. Una de las tonterías que siempre digo es que en una sociedad avanzada de verdad, los profesionales mejor remunerados, debido a lo que aportan a ésta, deberían de ser los médicos que nos salvan la vida y los profesores que forman a los futuros profesionales, sin estos últimos la actual sociedad avanzada en la que suponemos que vivimos sería inviable.

Por supuesto a Juan Llorens Morillo. Gracias por tu paciencia y por tu aguante con nosotros y gracias por la posibilidad de mandarnos a la bella isla a la que fuimos, gracias por enseñárnosla y por darnos la posibilidad de conocer que una sociedad mejor es posible.

Gracias a las Islas Aland y a la ciudad de Marieham, y a Suecia y Finlandia que de una manera u otra me abrieron los ojos por un lado, a pesar que debido a ello cada vez me siento más crítico y desesperanzado con el país en el que vivo.

Gracias a España, por su diversidad, por sus paisajes, sus monumentos y las personas. En una pena que un país capaz de ser pionero en trasplantes, en energías alternativas y en la que hay personas muy muy validas tenga los mandatarios que tiene, y que la lastran hasta la situación que sufre.

TABLA DE CONTENIDO

INTRODUCCIÓN

Actualmente es inconcebible un mundo sin informática. Todo, absolutamente todo está controlado por ordenadores: Fábricas de juguetes, fábricas de cerveza, fábricas de armas, archivos de la seguridad social, bancos y un sin fin de etcéteras que sería imposible de listar. Nuestro mundo cambiaría radicalmente si las máquinas desaparecieran.

Aun así el control total sigue siendo nuestro, del hombre, que somos aquellos que desarrollamos los controladores para esas máquinas. Esos expertos psicólogos de maquinas que se comunican con ellas, son los generalmente conocidos como informáticos, ese tipo con una imagen hacia el exterior de tipo raro que habla con las ordenadores (y por ende debe arreglarte el televisor e incluso el tostador).

El problema de los generalmente llamados informáticos, es que los ordenadores no son más que máquinas calculadoras con un increíble potencial, pero que únicamente entienden instrucciones básicas. Para ello se desarrollaron los lenguajes de programación: unos lenguajes que son “simples” de aprender por el ser humano y que mediante unos programas llamados compiladores pueden ser traducidos a esas instrucciones básicas que entiende el ordenador.

Ahora bien, como se comentaba anteriormente el mundo ha pasado a ser controlado de una manera u otra, o llamémoslo mejor tutorado, por las máquinas, todas las grandes empresas independientemente del sector que sea, tienen un departamento de informática, o en su lugar le confían a empresas expertas en ese ámbito, el de la informática, al que encomiendan toda la creación y mantenimiento de esos programas que deben hacerles más fácil su trabajo.

Muchas de estas empresas a las que se les contrata la creación y el mantenimiento de aplicaciones tienen un gran número de empleados que se encargan de crear estas aplicaciones. En muchas ocasiones estas empresas acometen proyectos de mucha similitud: imaginemos que a una empresa la contrata MAPFRE para hacer un sistema gestor de siniestros, y más adelante es REALE la que la contrata para el mismo cometido. Dicha empresa tiene la ventaja de tener un conocimiento previo funcional del problema, lo que va a facilitar la acometida del segundo proyecto. A nivel del desarrollo de la aplicación esta empresa también debería tener un conocimiento previo de lo que se debe implementar. A este punto se le llama reutilización de software, al proceso de volver a emplear unos conocimientos que ya tenemos en otros proyectos.

Dentro del heterogéneo mundo de los informáticos existen distintos puestos según sus aptitudes y el trabajo que llevan a cabo. Así no es el mismo tipo de informático el que va a una empresa a preguntar qué es lo que se quiere, que el que una vez sabido lo que se quiere lo plasma de manera adecuada y

formal a modo que lo pueda entender cualquier persona con unos mínimos conocimientos del tema, que aquella persona que se encarga de codificar el resultado de este documento.

Todos los proyectos empresariales deberían seguir una metodología definida, en la que como mínimo deberían existir tres pasos en este orden: Toma de requisito, Análisis y Diseño e Implementación. Estos tres pasos no son suficientes en sí mismos para constituir una metodología pero una metodología sin estos pasos será a todas luces incorrectas.

El gran problema es que actualmente y gracias a que todas las empresas priorizan el maximizar sus ingresos sin tener en cuenta ni siquiera que su producto cumpla lo solicitado, ni que se entregue en tiempo, ni la reputación de dicha empresa; en muchas ocasiones se encuentran proyectos en los que se salta el paso del análisis o diseño o como mínimo no se plasma en un documento que perdure en el tiempo y que sea reutilizable en el futuro para posteriores proyecto.

Como para casi todo en este mundo que tienen que ver con procesos industriales, el ser humano ha intentado crear metodologías o estándares para facilitar la comprensión por todo el mundo de estos procesos informáticos de desarrollo de Software. Dentro del punto comentado anteriormente de Análisis y Diseño, esta admitido como un estándar de facto la utilización de diagramas para comprender rápidamente la estructura de las aplicaciones, el metalenguaje actualmente más utilizado para este tipo de diagramas es el llamado UML (Unified Modeled Language).

Por otro lado existe la gran red de redes, Internet, lugar del que se pueden obtener miles y miles de soluciones a problemas que altruistamente han sido colgadas por desarrolladores que querían compartir sus conocimientos con el mundo. Estas soluciones suponen una fuente casi inagotable de conocimiento y en muchas ocasiones sirven de docencia para implementadores expertos que no dominan todo el espectro de la programación. Estos fragmentos de código o soluciones únicamente tienen un problema, obviamente la gente que los sube, no sigue el proceso formal de desarrollo con una metodología y cuelga además del código su información, normal ya que no es su cometido. Esto hace que sea necesario empaparse de este conocimiento y plasmarlo por parte del desarrollador que lo utiliza en caso de que quiera asumir ese ejemplo como parte de su conocimiento o el de su empresa para reutilizarlo en el futuro.

Por parte del departamento Knowledge Reuse de la universidad Carlos III se está desarrollando un proyecto piloto que sea capaz de, partiendo de un fragmento de código, obtener la mayor metainformación posible en UML de esta para poder plasmar de manera conceptual cómo funciona el código propuesto. Como es deducible este proyecto es amplísimo por lo que se le ha dotado a la aplicación resultante de la posibilidad de utilizar distintos indizadores, así es como se le llama a la librería capaz de obtener la metainformación y almacenarla de un fragmento de código. Obviamente es

prácticamente imposible crear una aplicación que sea capaz de “comprender” todos los lenguajes de programación que se utilizan actualmente en el mundo.

Con estas premisas parte este proyecto, cuya finalidad es la de crear un indizador que sea capaz de obtener la metainformación de las soluciones de Visual Studio .NET 2008 que hayan sido escritas en el lenguaje Visual Basic.

GLOSARIO

UML – Unified Modeled Language – Lenguaje unificado de modelado. Lenguaje gráfico utilizado para la especificación de características de sistemas.

nUNIT – Librería de pruebas para .net que permite la automatización de las mismas, pertenece al conjunto de pruebas llamadas de caja blanca.

nUML – Componente para la automatización de metainformación en UML. Este componente ha sido proporcionado por el departamento XX de la universidad Carlos II de Madrid

VB – Visual Basic – Lenguaje muy extendido actualmente.

IDE – Integrated Development Environment – Entorno Integrado de desarrollo

CLR – Common Language Runtime – Entorno común de ejecución de lenguajes

MSIL - Microsoft Intermediate Language – Lenguaje Intermedio de Microsoft

BCL – Basic Class Library – Librería de clases base.

API – Application Programming Interface – Interfaz de programación de aplicaciones

XMI- XMI o XML Metadata Interchange (XML de Intercambio de Metadatos) es una especificación para el Intercambio de Diagramas.

ESTADO DEL ARTE

En este apartado se lista las herramientas y conceptos que han servido como base al desarrollo de este proyecto. Se comentaran los aspectos fundamentales de estas herramientas y tecnologías sin las que este proyecto no habría sido posible. Se listaran también el marco donde se sitúa este proyecto, antecedentes del mismo y aplicación en la que se engloba.

Este punto es fundamental para comprender más tarde el origen y las herramientas que han permitido y originado el desarrollo de este proyecto.

Debido a la singularidad de este proyecto, este apartado será muy similar al apartado de herramientas ya que, para este caso se unen en demasía, ya que, ejemplos como el Visual Studio forman origen, parte y herramienta de este proyecto. Origen debido a que la gran extensión dentro del mundo del desarrollo de aplicaciones de este framework le convierte en una herramienta muy importante de automatizar ya que ayudaría a muchos desarrolladores, parte ya que como se comenta son las soluciones de Visual Studio las que se pretenden automatizar, y herramienta ya que es dentro de este framework, el de Visual Studio 2008, donde vamos a implementar el componente necesario para la automatización de soluciones.

.NET FRAMEWORK

Se trata de una plataforma de desarrollo de software creada por Microsoft y compuesta por un conjunto de bloques interconectados, diseñados todos ellos de manera homogénea permitiendo el desarrollo de aplicaciones para los sistemas operativos Windows que resultan fáciles de implementar, gestionar, implantar e integrar en sistemas en red.

Este framework pone a disposición del implementador un amplio conjunto de librerías, que solucionan muchos de los problemas recurrentes que suelen aparecer durante el desarrollo de la mayoría de las aplicaciones reduciendo la complejidad y ahorrando tiempo.

Las más importantes características de este entorno de trabajo son:

- **Independencia del lenguaje**: con .NET existe la posibilidad de desarrollar código para la plataforma en una gran variedad de lenguajes. Todos ellos generan al ser compilados un código intermedio conocido como MSIL (Microsoft Intermediate Language), que es lo que realmente se necesita a la hora de ejecutar un programa.
- **Interoperabilidad**: la plataforma .NET ofrece los mecanismos necesarios para acceder a las funcionalidades implementadas en programas que se encuentran fuera del entorno .NET, lo que amplía considerablemente las posibilidades del desarrollador a la hora de diseñar sus programas.
- **Portabilidad**: Quizá el punto más discutible ya que aunque resulta posible implementar la plataforma sobre otros soportes como Unix o Mac OSX, estas implementaciones son meramente didácticas, realmente las implementaciones comerciales de Microsoft sólo cubren Windows y Xbox360.

Desde que en Febrero del 2002 Microsoft liberará la primera versión del framework, llamado ".Net Framework 1.0", muchas han sido las versiones que han ido surgiendo hasta el día de hoy, siendo la última en ver la luz la versión 4.0.

Los dos componentes principales que configuran la plataforma .NET son: el llamado CRL (Common Language Runtime) y el llamado BCL (Base Classes Library), a continuación se comentan las principales características de estos:

CLR (ENTORNO COMUN DE EJECUCION PARA LENGUAJES)

Se trata de un entorno de desarrollo y de ejecución, que no depende de ningún lenguaje de programación en concreto, encargado de gestionar la ejecución de una aplicación en la plataforma .NET.

En el ámbito de .NET el código que es ejecutado por el CLR recibe el nombre de código manejado, mientras que el código no manejado es aquel que se ejecuta directamente sobre el sistema operativo.

El CLR mantiene activados normalmente una serie de servicios automáticos que supervisan la ejecución del código manejado. Ejemplo de ellos son:

- **Cargador de clases:** permite la carga en memoria de las clases.
- **Compilador MSIL a nativo**: el código máquina real que es ejecutado en la plataforma del cliente se obtiene compilando el código intermedio justo en el momento de la ejecución. Es este compilador el que obtiene las instrucciones nativas a ejecutar a partir del código MSIL generado con antelación.
- **Recolector de basura:** se encarga de liberar todas las zonas de memoria que siguen reservadas por una aplicación, aunque en realidad ya no están siendo utilizadas.
- **Motor de seguridad:** administra la seguridad del código que se ejecuta.

En definitiva este componente viene a ser lo mismo que la maquina virtual de java que se encarga de su ejecución.

Existen más pero no son relevantes para este proyecto.

BCL (LIBRERIA DE CLASES BASE)

Se trata de una librería que agrupa la mayor parte de las funcionalidades comunes a las que tienen acceso todos los lenguajes de la plataforma .NET. Está orientada a objetos y las clases que ofrece encapsulan las operaciones básicas que participan típicamente en el desarrollo de aplicaciones.

Esta librería facilita la programación para los desarrolladores gracias al manejo de por ejemplo estructuras de datos que se utilizan recurrentemente o el acceso a componentes del sistema operativo que en ocasiones son necesarias para el desarrollo de aplicaciones.

VISUAL STUDIO

Visual Studio es un entorno de programación integrado, también llamado IDE (Integrated development environment) creado por Microsoft.

Esta plataforma de desarrollo se basa en los sistemas operativos de la misma compañía. Actualmente se está introduciendo en el mercado la versión 2010.

La primera versión, llamada 6.0, se basaba en el modo normal de trabajo de Visual Studio antes de la introducción del framework .NET, es decir utilizaba la API Win32. Posteriores versiones se basan en el framework .net.

Este entorno de programación permite al implementador la utilización de varios lenguajes de programación, haciendo que todas las aplicaciones sean interconectables entre sí independiente del lenguaje en el que se hayan desarrollado inicialmente gracias al potencial del framework .NET, lo que facilita la programación y su utilización en red, además de la homogeneización de programadores que comenzaron a programar en distintos lenguajes.

Los lenguajes que actualmente acepta este IDE son:

- **VB** Los orígenes de este lenguaje datan de 1991, cuando fue creado por Alan Cooper. Debido a su antigüedad es un lenguaje muy extendido y conocido por infinidad de programadores, la tendencia por parte de Microsoft es el abandono de este lenguaje aunque aun esta soportado.
- **C++** Lenguaje creado a mediados de los 80 como sucesor de C. La principal evolución de este lenguaje es la inclusión de la orientación a objetos, lo que hace este lenguaje mucho más manejable y fácil de mantener, además de permitir afrontar problemas más complejos desde un punto de vista más sencillo.
- **C#** Como C++ fue el salto evolutivo de C, Microsoft creo este lenguaje como evolución de C++. Con gran similitud con Java, este lenguaje se desarrollado específicamente para ser utilizado bajo la plataforma .Net. Creado en 2001 pretende convertirse en el lenguaje oficial de Visual Studio.
- **ASP** Lenguaje desarrollado por Microsoft como extensión de IIS (Internet Information Services) para el desarrollo de aplicaciones Web de manera simple.

Los lenguajes descritos anteriormente no son los únicos pero si se han convertido en los más extendidos, a lo largo de las distintas versiones de Visual Studio se han ido introduciendo y eliminando distintos lenguajes:

- **J++**
- **J#**
- **F#**

Esta plataforma nos ofrece una extensa API (application programming interface) con la que podremos ahorrarnos una gran cantidad de tiempo al tener implementada en ésta operaciones para el manejo recurrente de estructuras muy utilizadas y acceso a componentes del sistema operativo, así como una gran cantidad de herramientas que sirven para mejorar la productividad y disminuir el estrés de los programadores.

La versión utilizada para este proyecto es Visual Studio 2008

ESTADAR DE COMPRESION .ZIP

Formato de compresión sin pérdida muy extendido en el mundo actual, su primera versión fue liberada allá por el 1989.

Una de las principales características de este formato de compresión es la de que actúa sobre cada uno de los archivos que se incluyen en el independientemente lo que facilita el acceso independiente a cada uno de ellos sin tener que descomprimir los demás archivos del mismo fichero.

En el siguiente grafico se ve el ratio de compresión de distintos formatos de compresión utilizados en la actualidad.

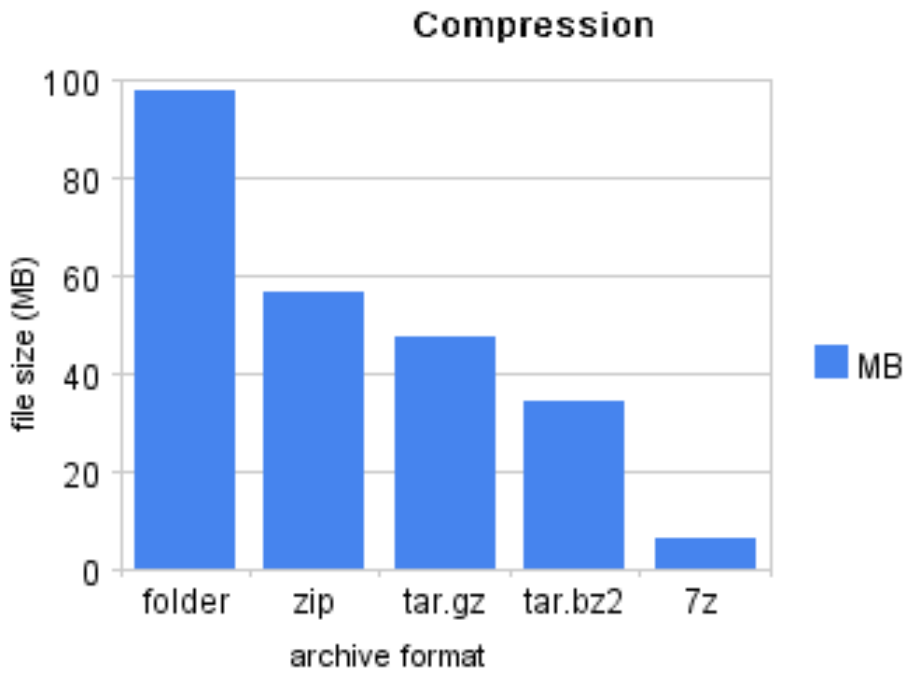

Ilustración 1: Relación de compresión en distintos formatos

Como se puede observar en este gráfico, quizás no sea el formato de compresión más eficiente, es decir, no es el que ahorra más espacio; pero su característica de descomprimir los archivos por separado y la gran extensión por el mundo digital de ese formato hace que sea el más interesante de utilizar para este proyecto.

UML

El Lenguaje unificado de modelado (de sus siglas en ingles Unified Modeled Language) es el lenguaje de modelado más utilizado actualmente del mundo. La intención de este lenguaje es la de especificar, visualizar y documentar cualquier sistema.

Para este cometido este lenguaje presenta distintos tipos de diagramas en los que se pueden apreciar distintos aspectos del sistema. Actualmente la versión más extendida es la UML 2.0 en la que los diagramas existentes son:

Diagramas de estructuras: Este tipo de diagrama muestra los elementos que deben existir en el sistema.

- **Diagrama de Clases.**
- **Diagramas de Componentes.**
- **Diagrama de Objetos.**
- **Diagrama de Estructura Compuesta.**
- **Diagrama de Despliegue.**
- **Diagrama de Paquetes.**

Diagramas de Comportamiento: Muestra lo que debe suceder en el sistema

- **Diagrama de Actividades.**
- **Diagrama de casos de Uso.**
- **Diagrama de Estados.**

Diagramas de Interacción: Con un cometido similar a los de comportamiento, muestran el flujo de control y de los datos entre los elementos que existen en el sistema

- **Diagrama de Secuencia.**
- **Diagrama de Comunicación.**
- **Diagrama de Tiempos.**
- **Diagrama de vista de Interacción.**

Este leguaje esta aceptado por la OMG (Object Management Group), por lo que su utilización se ha convertido prácticamente en un estándar dentro de las metodologías de desarrollo, especialmente cuando se va a enfocar el desarrollo en orientación a objetos. Aun así este lenguaje es válido para definir sistemas estructurados debido a la gran cantidad de tipos diagramas que lo componen.

NUML

nUML es una librería para manipular metamodelos descritos en UML 2.0, esta herramienta permite almacenar y manejar la metadata de los modelos UML.

Esta librería es la utilizada en el macrosistema en el que se engloba este proyecto por lo que se toma como elemento principal del mismo.

En puntos más avanzados del sistema se describirán los componentes del sistema y como se debe utilizar el mismo para poder reflejar la metainformación de las distintas soluciones.

NUNIT

Librería para automatizar las pruebas unitarias heredero de JUnit, para la plataforma de desarrollo .net. Existen varias extensiones para el testeo de Forms y aplicaciones web.

Las pruebas unitarias, también llamadas de caja blanca, cumplen con el comedido de probar la aplicación de la forma más detallista posible, el propósito es probar cada una de las funciones para validar que no se encontraran problemas en la aplicación, probando cada uno de los módulos de forma separada.

Existen varias características que se desean dentro de una completa prueba unitaria estas son:

- **Automatizable** – Se refiere a que no necesita de intervención manual por parte del desarrollador.
- **Completa** – Debe estar testeado la mayor parte del código posible para validar que no aparezcan problemas por no haberlo probado.
- **Repetibles** – Las pruebas deben poder ser reutilizadas para poder ser usadas a lo largo de todo el desarrollo
- **Independientes** – Las pruebas han de ser lo más independientes entre sí para no afectar unas a otras.

La utilización de las pruebas unitarias aporta al proyecto muchas ventajas ya que sirven como documentación, ya que con las pruebas unitarias podemos ver como se utilizan las funciones, simplifican la integración entre módulos ya que al estar probado cada uno de ellos descartamos problemas cuando la integración tiene problemas.

Estas pruebas además nos permiten la separación entre la interfaz y la implementación ya que ya no es necesario lanzar la aplicación y probarla totalmente para ver que esta funciona, podemos probar los distintos aspectos de la solución.

ENVDTE

Este espacio de nombres proporcionado por Microsoft facilita la automatización de soluciones .net.

Los Objetos EnvDTE se crean a raíz de una solución de Visual Studio y se basan en la descripción XML que estos proporcionan para obtener la máxima información posible del código de la solución sin tener que compilarla.

La búsqueda en sí de este componente constituyo gran parte de la ruptura de la tecnología de este proyecto.

Gracias a este componente este proyecto puede obtener la suficiente información como para poder reflejar la estructura del proyecto en UML a través de nUML.

Este componente ofrece todo tipo de funciones para describir la solución desde la que se creó el objeto.

OTRAS APLICACIONES

A continuación se dará una visión global del funcionamiento del indexador y su integración dentro de la aplicación matriz dentro de la cual se engloba el indizador a generar.

La aplicación dentro de la que se engloba consiste en un indizador global de soluciones. A continuación se expondrá de manera conceptual la forma en la que trabaja la aplicación:

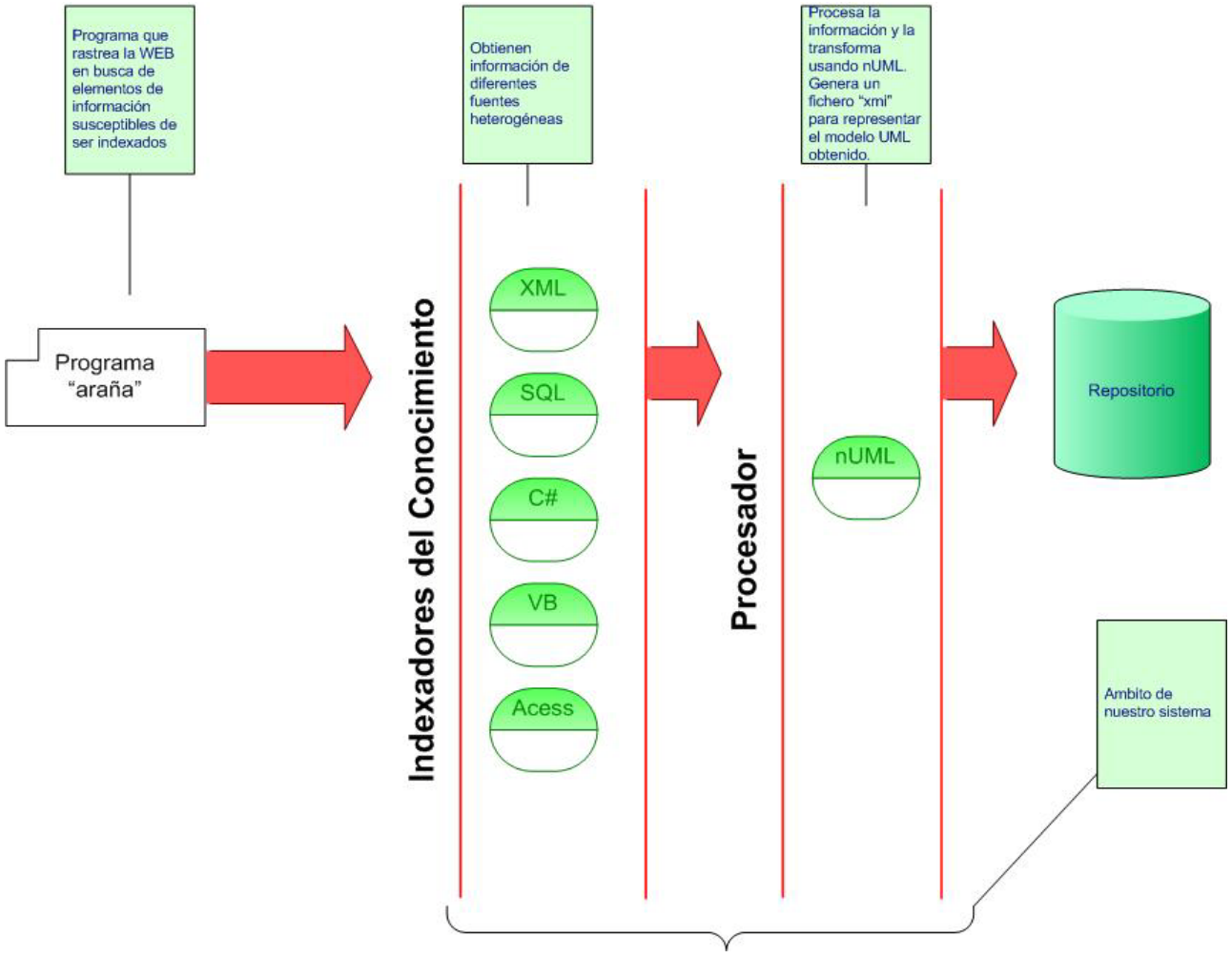

Ilustración 2: Esquema de aplicación superior

Se recuerda que el objetivo del proyecto es la creación de un sistema capaz de analizar las soluciones de Visual Studio contenidas en un fichero .zip y generar un modelo UML con la información obtenida de las mismas. En la figura anterior se muestra como queda encuadrado el indexador dentro del sistema matriz.

Se pueden identificar los siguientes elementos:

Programa "araña": Parte del sistema matriz que se encarga de rastrear la web en busca de soportes de información susceptible de ser indexados. En el caso de este proyecto deberá buscar ficheros .zip o soluciones .Net. Posteriormente el indexador VB determinará si estos ficheros contienen información válida, es decir soluciones de Visual Studio. El desarrollo de este subsistema queda fuera del ámbito del proyecto.

Indexadores del Conocimiento: Subsistemas que permiten analizar la información contenida en diferente soportes. Estos subsistemas pueden estar dedicados a una gran variedad de fuentes, como por ejemplo: ficheros xml, sql, acess. En el caso de este proyecto está asociado a los ficheros de código VB contenidos en las soluciones analizadas.

Procesador: Una vez indexada la información es necesario transformarla en un modelo UML, de manera que a partir de multitud de fuentes heterogéneas: xml, acess, squl, c# obtengamos una salida homogénea UML. De esta transformación se encarga una parte del indexador conocida con el nombre de procesador, que gracias al uso de nUML genera un fichero "xmi" para almacenar el modelo UML.

Repositorio: Base de datos donde se almacena todo el conocimiento indexado tras este proceso. La creación y gestión de este repositorio queda fuera del ámbito de este proyecto.

VISUAL BASIC

Lenguaje creado en 1991 por Alan Cooper para Microsoft. Basado en Basic se trata de un lenguaje integrado dentro de un IDE (Integrated development environment, entorno integrado de desarrollo), es decir viene empaquetado dentro de una aplicación con un editor, un compilador y un constructor de interfaz grafica.

Este Entorno nos proporciona muchos controles gráficos para la rápida creación de aplicaciones visuales de modo que el desarrollador no tenga que ser un experto maquetador ni tiene por qué tener un extenso conocimiento en programación de controles para Windows.

Es un lenguaje de los llamados RAD o de desarrollo rápido de aplicaciones, lo que como su nombre indica nos permite crear rápidamente aplicaciones complejas para Windows.

Una de las principales características de este lenguaje y lo que le ha hecho ser tan conocido y propagado es su rápida curva de aprendizaje, no es un lenguaje complejo, además alguna de sus características facilitan este lenguaje: No admite el manejo de memoria dinámicamente, esto que para algunos tipos de programación (kernels por ejemplo) le resta potencia, sin embargo facilita su aprendizaje ya que el programador no tiene que estar pendiente de la memoria que utiliza.

Veamos cuales están consideradas sus principales ventajas e inconvenientes:

VENTAJAS

- Es un lenguaje RAD.
- Posee una curva de aprendizaje muy rápida.
- Integra el diseño e implementación de formularios de Windows.
- Permite usar con suma facilidad la plataforma de los sistemas Windows, dado que tiene acceso prácticamente total a la API de Windows, incluidas librerías actuales.
- El código en Visual Basic es fácilmente migrable a otros lenguajes.
- Es un lenguaje muy extendido, por lo que resulta fácil encontrar información, documentación y fuentes para los proyectos.
- Fácilmente extensible mediante librerías DLL y componentes ActiveX de otros lenguajes.
- Posibilidad de añadir soporte para ejecución de scripts, VBScript o JScript, en las aplicaciones mediante Microsoft Script Control.
- Acceso a la API multimedia de DirectX (versiones 7 y 8). También está disponible, de forma no oficial, un componente para trabajar con OpenGL 1.1: VBOpenGL type library

- Existe una versión integrada en las aplicaciones de Office, versiones tanto Windows como Mac, que permite programar macros para extender y automatizar funcionalidades en documentos como por ejemplo una hoja de cálculo de Excel o una base de datos Access (VBA).
- Es un entorno perfecto para realizar pequeños prototipos rápidos de ideas.

INCONVENIENTES

- Sin soporte oficial de Microsoft desde el 4 de abril de 2008 (aunque existe mucha documentación disponible en el sitio de MSDN incluidas descargas de Service Packs, paquetes de dependencias mínimas y similares en el sitio web de Microsoft).
- No es multiplataforma (sin embargo se pueden usar emuladores e intérpretes para correrlos en otras plataformas).
- Por defecto permite la programación sin declaración de variables, (que puede ser sencillamente corregida escribiendo la frase Option Explicit en el encabezado de cada módulo de código, en cuyo caso será menester declarar todas las variables a utilizar, lo que a la postre genera código más estable y seguro).
- No permite programación a bajo nivel ni incrustar secciones de código en ASM.
- Sólo soporta librerías dinámicas (DLL) que usen la convención de llamadas _stdcall y componentes y librerías ActiveX.
- Es un lenguaje basado en objetos pero no implementa por completo la filosofía de Orientación a Objetos.
- No permite nombres de espacio.
- No soporta el puntero a memoria salvo en algunas acciones concretas, como por ejemplo pasar la dirección de memoria de una función como argumento (operador AddressOf).
- No soporta tratamiento de procesos como parte del lenguaje.
- No incluye operadores de desplazamiento de bits como parte del lenguaje.
- No permite el manejo de memoria dinámica, punteros, etc. como parte del lenguaje.
- No controla todos los errores de conversión de tipos dado que en muchas ocasiones hace conversiones al vuelo (sobre todo al usar variables de tipo Variant).
- Aunque existen opciones avanzadas en el compilador para desactivar los controladores de desbordamiento de enteros o las comprobaciones de límites en matrices entre otros (presumiblemente para optimizar y lograr algo de rendimiento) no es seguro del todo dado que hay más posibilidades de generar una excepción grave no controlada por el intérprete (y por consiguiente del programador) o un memory leak haciendo el programa altamente inestable e impredecible.
- No tiene instrucciones de preprocesamiento.
- El tratamiento de mensajes de Windows es básico e indirecto.

- La gran gama de controles incorporados son, sin embargo en algunos casos, muy generales, lo que lleva a tener que reprogramar nuevos controles para una necesidad concreta de la aplicación. Esto cambia radicalmente en Visual Basic .NET donde es posible reprogramar y mejorar o reutilizar los controles existentes.
- El depurador no es demasiado flexible ni cómodo en ciertas situaciones.
- Los controles personalizados no mejoran la potencia de la API de Windows, y en determinados casos acudir a ésta será el único modo de conseguir el control personalizado deseado.
- No soporta correctamente la programación multihilo haciendo muy inestable su ejecución inclusive desde el propio entorno de desarrollo.
- Su fuerte dependencia de librerías y componentes ActiveX que requieren de privilegios de administrador para poder instalar las aplicaciones (existen opciones de terceras aplicaciones para generar ejecutables independientes que embeben las dependencias en el propio código del ejecutable, por ejemplo: Molebox o Thinstall/VMware Thinapp).

OBJETIVOS, PROBLEMAS Y SOLUCIONES

Se listarán en este punto los distintos problemas que se han tenido que abordar a lo largo del proceso creativo de este proyecto.

Se recuerda que el objetivo final de este proyecto se basa en la creación de un indizador que pasándole como entrada una solución .NET escrita en VB en la edición 2008 del Visual Studio, o en su defecto, un archivo .ZIP que contenga soluciones como las anteriormente descritas, obtenga toda la metainformación de dicha solución en lenguaje UML. Para ello se debe utilizar el componente proporcionado nUML, mediante el cual se podrá incluir toda la información relevante para especificar dichas aplicaciones en un archivo xmi.

DESCOMPRESION DE ARCHIVOS .ZIP

OBJETIVO

Se desea descomprimir un archivo .Zip para obtener todas las soluciones .net que haya dentro de él escritas en Visual Basic.

PROBLEMA

Sería recomendable para cumplir esta necesidad, que no fuese necesario tener instalado en el servidor en el que se instale el componente, el programa WinZip, herramienta que mayoritariamente se emplea en el mundo actual para la descompresión de archivos de extensión .ZIP, existen otras aplicaciones que cumplen con dicho objetivo, pero en muchas ocasiones se trata de programas de pago.

Como en la mayoría de aplicaciones comerciales, es un punto importante de este proyecto el minimizar el costo del mismo.

SOLUCIONES

Se ha buscado en la red alguna librería que cumpla con dicho objetivo, es decir, la descompresión de archivos .Zip y que no sea de tipo propietario.

Navegando se pueden encontrar muchos ejemplos de librerías subidas por programadores altruistas que sin ánimo de lucro buscan compartir sus conocimientos con otros programadores.

En este caso se ha utilizado la librería IONic.ZIP que proporciona los mecanismos necesarios para este proyecto.

OBTENCION DE METAINFORMACION DE LA SOLUCIÓN .NET

OBJETIVO

Se desea obtener la máxima información posible de una solución .net, con el objetivo de poder almacenar dicha información para su futura reutilización cuando se afronte algún tipo de proyecto con una estructura similar.

PROBLEMA

El gran problema de este proyecto, la automatización de soluciones .NET, se basa en este punto: Como, partiendo de una solución .NET se puede obtener la información que esta tiene.

Este es el gran trabajo de investigación y el que más tiempo implica.

SOLUCIONES

Finalmente se ha encontrado dentro de la librería de clases del framework de .net un componente llamado EnvDTE que nos permite abrir una solución .NET e ir iterando por los distintos elementos que esta contiene, gracias a ello y a un análisis de los elementos que nos va otorgando podemos obtener los distintos elementos que componen la solución y son validos para plasmar en UML.

Para la utilización de este componente se necesita lanzar un Visual Studio 2008 en background.

UTILIZACIÓN DE LA LIBRERIA NUML

OBJETIVO

La metainformación que se obtenga de las soluciones debe ser devuelta a través de la librería proporcionada nUML, esta proporciona las herramientas necesarias para una vez descrito el sistema devuelva un fichero xmi con dicha metainformación.

Junto a la librería se facilito un ejemplo de utilización que es el utilizado como base para su utilización.

PROBLEMA

La forma en la que se obtiene la metainformación por parte de la aplicación que va a utilizar este componente es mediante las funciones del paquete nUML. Por ello es necesario familiarizarse con dicho paquete y utilizarlo para incluir la metainformación.

SOLUCIONES

Gracias al ejemplo proporcionado y a la gran disposición por parte de la gente del departamento encargado de desarrollar la aplicación que utilizara este componente, se ha podido obtener toda la información disponible desde una solución .net. Más adelante se describirá en detalle como se ha utilizado dicho paquete.

METODOLOGIA DEL DESARROLLO

En este punto se analiza la metodología empleada durante el proceso creativo y de implementación de este proyecto.

Este como la mayoría de los proyectos se basa en la investigación por lo que hasta una vez demostrada su viabilidad y por lo tanto hasta ese punto no se pudo utilizar una metodología clara.

Una vez se encontraron los mecanismos que permitían el desarrollo de la solución para el problema propuesto, es decir, una vez vencida la tecnología, se ha aplicado una metodología clásica en cascada, que se ha ido transformando a lo largo del desarrollo en una metodología de cascada incremental, es decir se hace un desarrollo clásico en cascada, pero se siguen añadiendo funcionalidades según el cliente facilita nuevos requerimientos.

Esta metodología consiste en los siguientes pasos clásicos del desarrollo en cascada:

- **Análisis** – Primera fase del desarrollo, en este punto se extraen y estudian las necesidades y los requerimientos del usuario para poder poner en claro lo que se quiere.
- **Diseño** – Se estudia y descomponen las necesidades solicitadas por el usuario en la anterior etapa; y se proyecta el cómo se debe llevar a cabo el desarrollo de ese sistema: Que tecnologías y arquitecturas desarrollar para satisfacer el problema.
- **Implementación** – Una vez creado el sistema en esta fase se debe llevar a cabo la creación del sistema funcional, codificándolo como se sugirió en el anterior punto. En este punto se deben realizar determinadas pruebas, en concreto las unitarias, en otros puntos de este documento se describe en qué consisten estas pruebas.
- **Pruebas** – Una vez que esta codificada la solución se debe validar que dicha implementación es correcta y cumple con todos los objetivos solicitados por el usuario
- **Implantación** – La solución propuesta una vez probada se pone en producción, es decir se instala en las maquinas destinadas a tal efecto. Este paso no se ha afrontado dentro de este proyecto ya que al tratarse de un proyecto universitario pasara por un control previo de calidad antes de que sea empleado por parte de los clientes finales.

ENTORNO DE TRABAJO

En este punto se va a pasar a describir las herramientas utilizadas en este proyecto. Este punto va a ser muy similar al apartado de estado del arte ya que las propias herramientas que se han utilizado son la base para este sistema.

MICROSOFT OFFICE

Se utilizaran distintas herramientas de este conglomerado de aplicaciones, a saber:

MICROSOFT OFFICE WORD

Este editor de texto es posiblemente el más utilizado actualmente del mundo. Está diseñado para ser utilizado bajo los sistemas operativos de Windows. Es un editor fácil de utilizar y cuyos formatos de guardado están básicamente aceptados como estándar en el mundo informático empresarial actual.

Se ha utilizado esta herramienta para generar la documentación del proyecto.

MICROSOFT OFFICE VISIO

Editor de diagramas de la casa Microsoft especialmente diseñado para la creación de información descriptiva visual de sistemas.

Se ha utilizado dentro de este proyecto para la generación de los distintos diagramas UML que son presentados en esta documentación para describir el sistema implementado.

MICROSOFT OFFICE PROYECT

Herramienta utilizada para la planificación de tiempos de proyectos, creada para ser utilizada en sistemas Windows.

Se ha utilizado dentro de este proyecto para la generación del diagrama de Gantt que describe los tiempos necesarios para la implementación del proyecto, describiendo del mismo modo los distintos pasos y etapas por los que ha transcurrido el proyecto.

UML

A lo largo de este proyecto unas de las siglas más utilizadas serán UML. Este lenguaje de Modelado Unificado es un concepto clave dentro del proyecto.

El objetivo, repetimos una vez más, es la creación de una librería que automatice el volcado descriptivo de las soluciones .net a UML a través de nUML. Por lo tanto este lenguaje es base, herramienta y salida de este proyecto.

Como ya comentábamos en el estado del arte este lenguaje descriptivo consta de varios tipos de diagramas.

MICROSOFT VISUAL STUDIO .NET 2008

Al igual que UML, Visual Studio, dentro de este proyecto es parte y herramienta.

Es parte ya que la entrada de la librería será un proyecto de Visual Studio. Es herramienta ya que esta librería estará desarrollada en este Framework.

Visual Studio es un Framework de desarrollo creado especialmente para los sistemas operativos Windows.

Actualmente utiliza el Framework 3.5 pero se puede utilizar para crear aplicaciones de Frameworks anteriores. Por otra parte se ha lanzado la versión 2010 que se basa en el Framework 4.0.

INDIZADOR DE SOLUCIONES VB.NET

Veamos ahora el análisis y el diseño de la solución que se ha solicitado para dar por realizado este proyecto.

Es importante poner en conocimiento que el resultado de este proyecto no es una aplicación autónoma y autoejecutable si no un modulo para una aplicación mayor y más extensa que gestiona la automatización de soluciones dentro de un mundo heterogéneo de lenguajes y aplicaciones.

ANÁLISIS

A continuación se presentan los escenarios básicos de la librería y los requisitos solicitados por parte del "cliente" para la realización de la funcionalidad.

El funcionamiento básico sería:

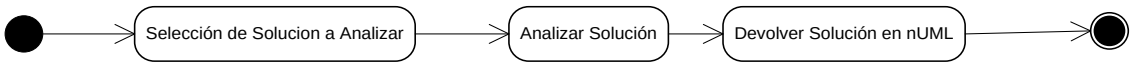

Ilustración 3: Escenario básico de la libreria

Los requisitos del sistema vienen definidos no tanto por los casos de uso de funcionamiento del sistema sino que han sido trasladados por parte del "cliente".

ENTORNO OPERACIONAL

Se describirá ahora el entorno en el que se debe ejecutar el componente proporcionado para su correcto funcionamiento.

Esta librería no tiene sentido en caso en que no sea utilizado dentro de un sistema más grande de obtención de metainformación. Este macrosistema ha sido desarrollado por el departamento "Knowledge Reuse" de la universidad Carlos III.

Esta DLL se incrusta dentro de este sistema para analizar las soluciones de Visual Studio escritas en VB.

Para que este componente funcione se debe tener instalado en la misma máquina el programa Visual Studio 2008 ya que utiliza esta aplicación para la obtención de información.

ESPECIFICACIÓN DE ESCENARIOS BÁSICOS Y CASOS DE USO

Dado que el programa desarrollado de indexación para soluciones VB pertenece a un sistema mucho más amplio y complejo y su existencia está ligado a su correcta integración con este, los casos de uso y escenario básicos no son demasiado numerosos quedado reducidos a las funcionalidad que la aplicación debe cumplir para el sistema matriz.

Aún así se han identificado los siguientes escenarios básicos y casos de uso de la aplicación que nos permiten abstraer la interacción de nuestro indexador con el usuario.

ESCENARIOS BÁSICOS.

Este es el escenario principal de la aplicación.

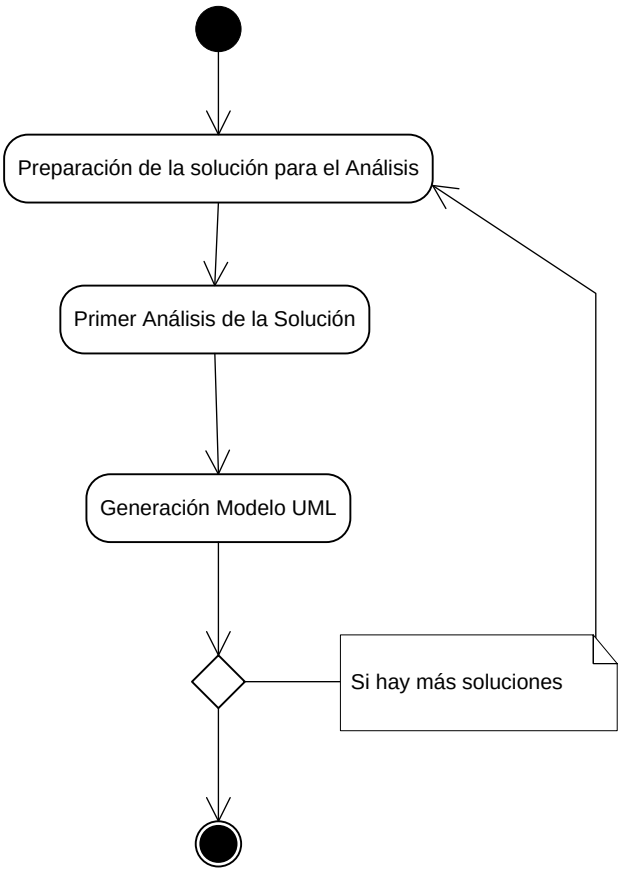

Ilustración 4: Escenario principal de la aplicación

En el escenario básico de la aplicación que se describe en la figura anterior se pueden identificar tres sub-escenarios:

- **Preparación de la solución para el análisis***:* Corresponde con el proceso de selección del fichero .zip que contendrá las soluciones a analizar, la comprobación de que existen soluciones válidas dentro de ese fichero y su posterior descompresión.

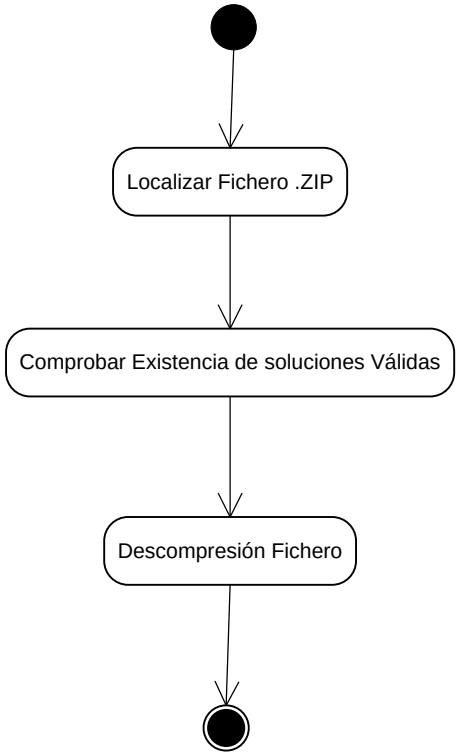

Ilustración 5: Preparación de la solución

- **Primer análisis de la solución***:* Una vez descomprimido el fichero .zip e identificadas las soluciones se procede a la fase de análisis de las mismas. En esta fase es cuando se identifican todos los elementos de código que aportan información relevante sobre la solución y que en la siguiente fase generará el metamodelo nUML. Posteriormente se enumerarán en detalle cuales son estos elementos de código. Para ello se abre la solución utilizando la aplicación Visual Studio pero en background, de manera que aunque VS está siendo ejecutado la interfaz visual del mismo no está visible. Una vez abierta la solución utilizamos la biblioteca COM para poder identificar los elementos de código presentes en la solución. Al final de esta fase obtendremos un pre-modelo con los elementos de código identificados.

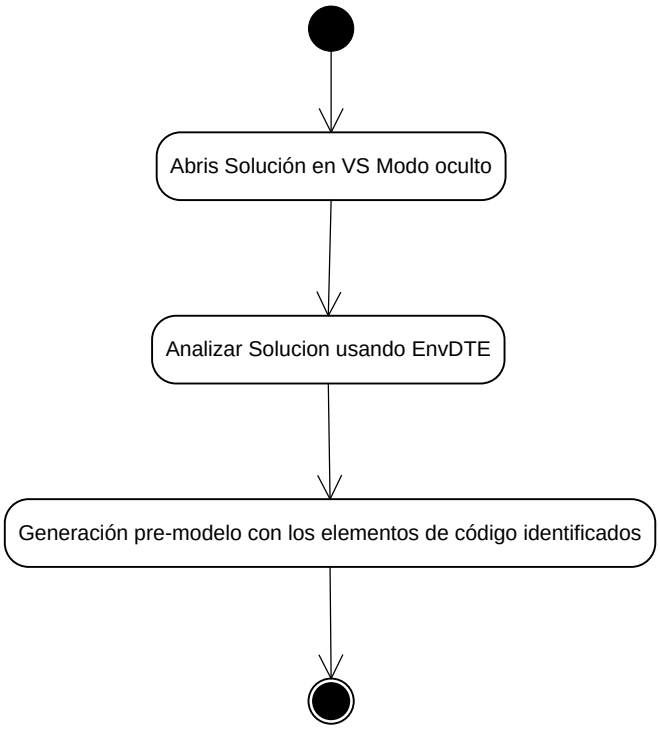

Ilustración 6: Análisis de la solución

- **Generación del modelo nUML:** Una vez que se han identificado los elementos de código que componen la solución, se debe transformar estos datos en información. Para ello, y partiendo del pre-modelo generado en la fase anterior, usamos la biblioteca nUML para transformar este pre-modelo en un metamodelo nUML.

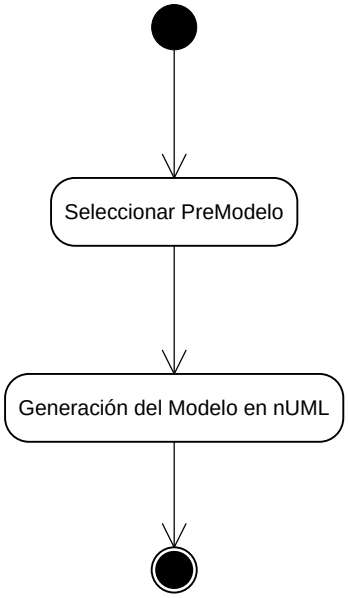

Ilustración 7: Generación de la metainformación

DIAGRAMA DE CASOS DE USO

A partir de los escenarios básicos identificados en el paso anterior se pueden definir el siguiente caso de uso.

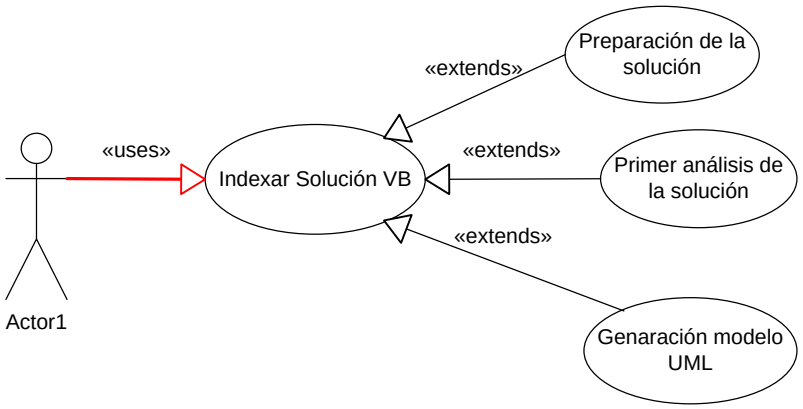

Ilustración 8: Diagrama básico de caso de uso

DESCRIPCIÓN TEXTUAL DE LOS CASOS DE USO

En este apartado se describen el caso de uso identificado en el apartado anterior. Con el fin de estructurar su análisis se han establecido, para cada uno de ellos, los siguientes atributos:

- **Identificador:** Se forma combinando las letras CU con un dígito secuencial-
- **Actores:** actores que participan en el caso de uso
- **Objetivo:** Finalidad del caso de uso, es decir lo que se persigue con la ejecución de este caso de uso
- **Precondiciones:** Descripción del estado del sistema antes de la ejecución del caso de uso.
- **Poscondiciones:** Descripción del estado del sistema después de la ejecución del caso de uso.
- **Escenario básico:** Secuencia de acciones que se enlazan durante la ejecución del caso de uso.

CU-01	
Nombre	Indexación soluciones.
Actores	Usuario.
Objetivo	Analizar e identificar los elementos de código recogidos dentro de soluciones VB y generar un metamodelo nUML
Precondiciones	Debe existir un fichero .zip con la soluciones.
Poscondiciones	Se generará un metamodelo nUML (información contenida en un fichero xmi) por cada solución encontrada en el fichero .zip.
Escenario básico	Seleccionar fichero .zip Analizar fichero .zip identificando la existencia de soluciones válidas Descomprimir fichero .zip Analizar e identificar los elementos de código de cada solución generando un pre-modelo. Transformar los pre-modelos del paso anterior en un metamodelo nUML.

REQUISITOS DE USUARIO

A continuación se enumeran los requisitos de usuario asociados a nuestra aplicación. Cada requisito representa “una condición o una funcionalidad exigidas por el usuario para resolver un problema o para conseguir un objetivo”, por lo que según esta definición, es posible clasificar a su vez los requisitos en dos categorías: requisitos de capacidad y requisitos de restricción.

Para cada uno se han fijado además los siguientes atributos:

- **Identificador de requisito**: cada requisito debe ser identificable unívocamente. Para los requisitos de capacidad se utiliza el formato RUC-XX y para los de restricción RUR-XX (siendo XX un número natural de dos cifras).
- **Necesidad**: los requisitos pueden clasificarse en esenciales, opcionales y convenientes según su importancia.
- **Estabilidad**: un requisito que no es estable presenta una gran dependencia de las fases posteriores del proceso de desarrollo y tiene una alta probabilidad de ser modificado en el futuro según se vaya profundizando en dichas fases.
- **Prioridad**: este atributo establece en qué momento respecto al resto de requisitos debería estar disponible la parte del sistema a la que se refiere el requisito.
- **Fuente:** informa acerca del origen del requisito**.**

REQUISITOS DE CAPACIDAD

RUC-01			
Descripción	La aplicación recibirá como fichero de entrada un fichero comprimido con extensión .zip que contendrá las soluciones a analizar.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-02			
Descripción	La aplicación utilizará Visual Studio para leer las soluciones escritas en lenguaje VB encontradas en el fichero .zip.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-03			
Descripción	Por cada solución se debe extraer la máxima información posible.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-04			
Descripción	El objetivo final de la aplicación es generar un meta-modelo nUML con la información indexada.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-05			
Descripción	Si en el fichero .zip de entrada existen varias soluciones se deberán analizar todas y cada una de ellas de forma independiente.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-06			
Descripción	Por cada solución encontrada se generará un meta-modelo UML contenido en un fichero xmi .		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

REQUISITOS DE RESTRICCIÓN

IDENTIFICADOR RUR-01			
Descripción	La DLL que resulte tiene que implementar la interfaz nUML. Transform proporcionado por el departamento Knowledge Reuse de la universidad Carlos III		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

IDENTIFICADOR RUR-02			
Descripción	El tipo del meta-modelo que se debe devolver es de tipo UML y está contenido en un fichero xmi		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

IDENTIFICADOR RUR-03			
Descripción	La DLL resultado debe estar implementada en la plataforma .net		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

IDENTIFICADOR RUR-04			
Descripción	La DLL resultado ejecutarse en entorno Windows.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

REQUISITOS SOFTWARE

Una vez identificados todos los requisitos de usuario de nuestra aplicación es posible enumerar los requisitos software que están reflejados en ellos. Los requisitos software nos dan una definición exacta de nuestra aplicación y son el punto de partida para las posteriores fases de análisis y diseño.

Para su descripción se ha utilizado la misma plantilla de atributos que en el caso de los requisitos de usuario.

El formato de los identificadores en este caso es diferente también: los requisitos de software funcionales utilizan **RSF-XX**, mientras que los no funcionales usan **RSN-XX**.

REQUISITOS FUNCIONALES

IDENTIFICADOR: RSF-01			
Descripción	Se recibirá un path que contenga la localización del fichero .zip.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-01		

IDENTIFICADOR: RSF-02			
Descripción	Es sistema debe realizar las descompresión del fichero .zip.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-01		

IDENTIFICADOR: RSF-03			
Descripción	Antes de realizar la descompresión se debe comprobar que existen soluciones válidas. Si no es así no se realiza la descompresión y la ejecución del la aplicación se da por finalizada.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-01		

IDENTIFICADOR: RSF-04			
Descripción	Si existen varias soluciones cada una se analizara de forma independiente.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-05		

IDENTIFICADOR: RSF-05			
Descripción	Cada solución encontrada se abrirá en la instancia de Visual Studio 2008 creada.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-02, RUC-05		

IDENTIFICADOR: RSF-06			
Descripción	No existirán simultáneamente varias instancias de Visual Studio 2008 ejecutándose al mismo tiempo. Solo se creara una instancia de Visual Studio 2008 sobre la que se irán cargando las soluciones.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-02, RUC-05		

IDENTIFICADOR: RSF-07			
Descripción	Se deben *matar* todas las instancias de Visual Studio 2008 que se hayan abierto para analizar una solución. Teniendo especial cuidado en el caso de que se haya producido una excepción durante la ejecución.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-02		

IDENTIFICADOR: RSF-08			
Descripción	Visual Studio 2008 se ejecutará de forma oculta de manera que aunque el programa se está ejecutando y tenga cargada una solución el usuario no vera la interfaz de Visual Studio 2008.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-02		

IDENTIFICADOR: RSF-09			
Descripción	Por cada solución se deben obtener todas las clases que existan		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-10			
Descripción	Por cada solución se deben obtener todos los interfaces.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-11			
Descripción	De cada clase se debe obtener: Nombre Visibilidad Si es abstracta o no Comentarios		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-12			
Descripción	Por cada clase se tiene que obtener todos sus atributos.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-13			
Descripción	Por cada clase se tiene que obtener todos sus métodos.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-14			
Descripción	Por cada clase se tiene que obtener todas las dependencias existan.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-15			
Descripción	Por cada clase se tiene que obtener todas las asociaciones que existan.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-16			
Descripción	Por cada clase se tiene que obtener todas las generalizaciones que existan.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-17			
Descripción	De cada interfaz se debe obtener: Nombre Visibilidad Comentarios existan		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-18			
Descripción	Por cada interfaz se deben obtener todos sus métodos.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-19			
Descripción	Por cada interfaz se debe obtener todas sus dependencias.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-20			
Descripción	En cada atributo se debe identificar: Nombre Tipo Visibilidad Comentarios		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-21			
Descripción	En cada método se debe identificar: Nombre del método Tipo de método (si es constructor o no) Si está sobrecargado Visibilidad Comentarios Valor de retorno Parámetros		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-22			
Descripción	En cada parámetro se debe identificar: Nombre Tipo		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-23			
Descripción	Con toda la información especificada en los requisitos: RSF09, RSF10, RSF11, RSF12, RSF13, RSF14, RSF15, RSF16, RSF17, RSF18, RSF19, RSF20, RSF21 y RSF22 se debe generar un meta-modelo nUML que la recoja.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-04		

IDENTIFICADOR: RSF-24			
Descripción	Se generarán tantos modelos nUML de salida como soluciones se hayan analizado		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-06		

REQUISITOS NO FUNCIONALES

IDENTIFICADOR: RNF-01			
Descripción	El código de nuestra aplicación debe implementar la interfaz nUML. Transform.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-01		

IDENTIFICADOR: RNF-02			
Descripción	Se devolverá un meta-modelo de tipo nUML		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-02		

IDENTIFICADOR: RNF-03			
Descripción	Se utilizará .NET como plataforma de desarrollo.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-03		

IDENTIFICADOR: RNF-04			
Descripción	Se utilizará Visual Studio 2008 como IDE de desarrollo.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-03		

IDENTIFICADOR: RNF-05			
Descripción	Windows será el entorno de ejecución de nuestra aplicación.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-04		

TRAZABILIDAD DE REQUISITOS

Tabla 1: Matriz de trazabilidad de Requisitos

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUR-01	RUR-02	RUR-03	RUR-04
RSF-01	•									
RSF-02	•									
RSF-03	•									
RSF-04					•					
RSF-05		•			•					
RSF-06		•			•					
RSF-07		•								
RSF-08		•								
RSF-09			•							
RSF-10			•							
RSF-11			•							
RSF-12			•							

RSF-13			•							
RSF-14			•							
RSF-15			•							
RSF-16			•							
RSF-17			•							
RSF-18			•							
RSF-19			•							
RSF-20			•							
RSF-21			•							
RSF-22			•							
RSF-23				•						
RSF-24						•				
RSN-01							•			
RSN-02								•		
RSN-03									•	

RSN-04									•	
RSN-05										•

DISEÑO

DISEÑO ARQUITECTONICO

A continuación se muestra cómo se integra el componente que se proporcionará, dentro de la arquitectura global de la aplicación:

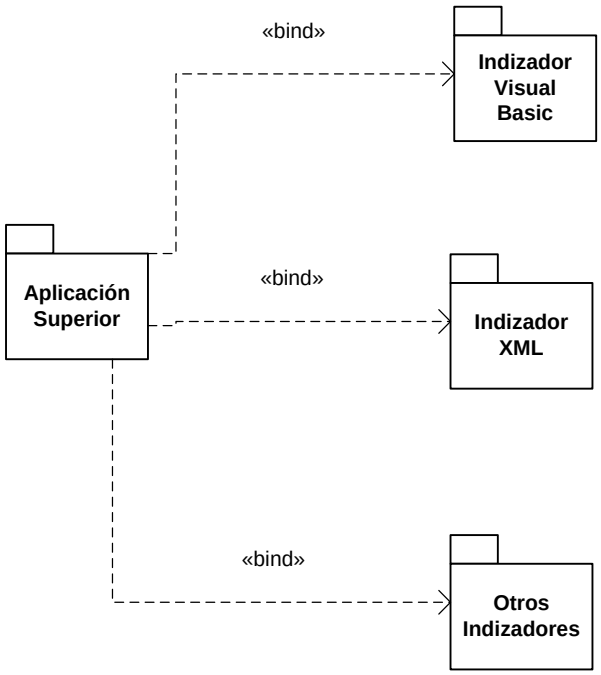

Ilustración 9: Diseño arquitectónico

Como se observa en el diagrama, el presente proyecto se basa en la creación del indizador de Visual Basic. Este componente no funciona por si solo si no que debe encuadrarse dentro del marco de trabajo de una aplicación superior. Para una mejor comprensión se describirá cada uno de los componentes del diagrama:

- **Aplicación superior.** Esta aplicación controla la gestión de los threads encargados de ir obteniendo la información de las distintas soluciones
- **Indizador de Visual Basic.** Componente resultante de este proyecto.
- **Indizador XML.** Ejemplo de indizador existente dentro del marco de trabajo. En este caso pasando como origen un archivo XML, el componente debería devolver la metainformación en nUML. Contenido en un fichero xmi
- **Otros Indizadores.** Debido al diseño de la aplicación superior, este permite la integración de nuevos indizadores de manera que se pueda indexar la mayor información posible que se puede encontrar dentro del heterogéneo mundo de aplicaciones informáticas.

ARQUITECTURA INTERNA

Se listara en este punto el diseño interno del componente de manera que sea lo más fácil posible de implementar siguiendo estas indicaciones. Para ello vamos a dividir este apartado en tres grandes puntos, los que a su vez podemos considerar como los tres problemas básicos del proyecto, estos son:

- **Descompresión del fichero .ZIP**
- **Lanzamiento de Visual Studio 2008 en background desde código**
- **Utilización del componente EnvDTE**
- **Utilización de la librería nUML**

Además de estos puntos se va a incluir previamente un punto comentando los elementos que se van a detectar de las soluciones tras haber realizado el análisis en los anteriores puntos.

Con el objetivo de poder indexar la información presente en una solución de Visual Studio, teniendo además la dificultad añadida de que las soluciones fuentes vienen comprimidas en un fichero .zip, se debe trabajar con diferentes tecnologías que de forma combinada nos permita alcanzar nuestro objetivo.

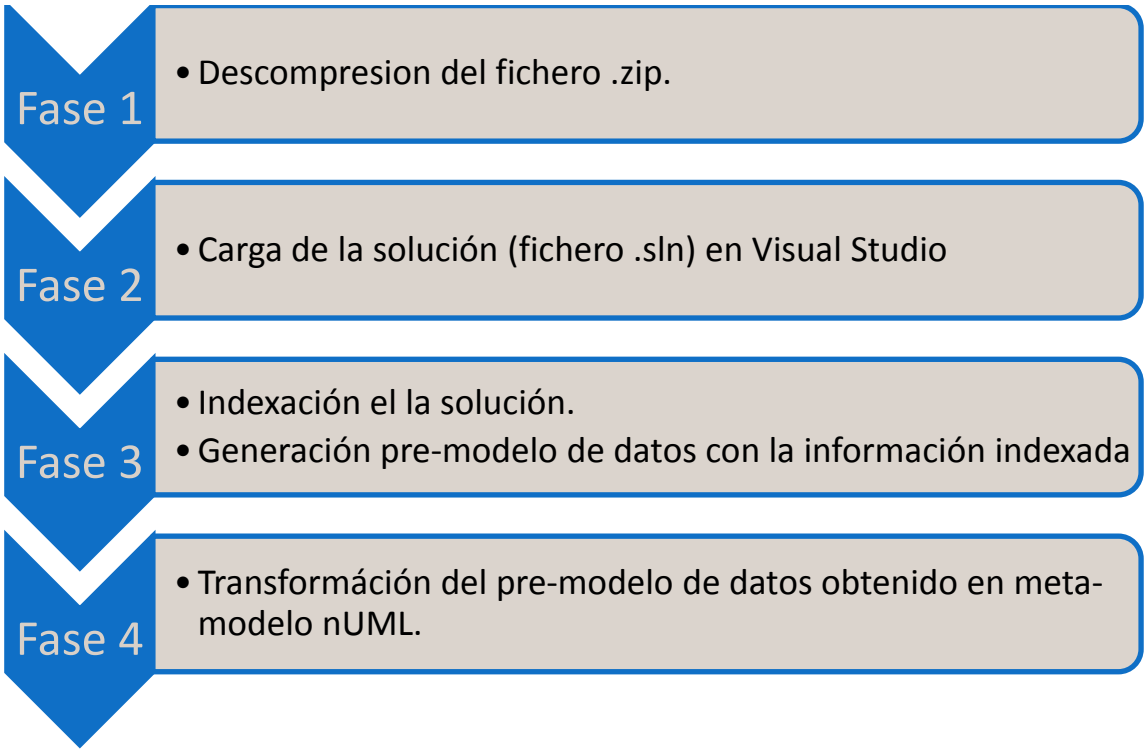

Ilustración 10: Esquema básico de funcionamiento

En esta se pueden identificar las cuatro fases principales de ejecución de la aplicación que van asociadas con los cuatro principales bloques de diseño que hemos tenido que abordar durante la realización del sistema. Si bien toda la aplicación ha sido desarrollada bajo .NET para cada uno de los bloque de diseño ha sido necesario el uso de una o varias bibliotecas que han jugado un papel clave y que nos han permitido poder alcanzar los objetivos marcados. En el siguiente gráfico se muestran asociadas a las fases las tecnologías clave aplicadas.

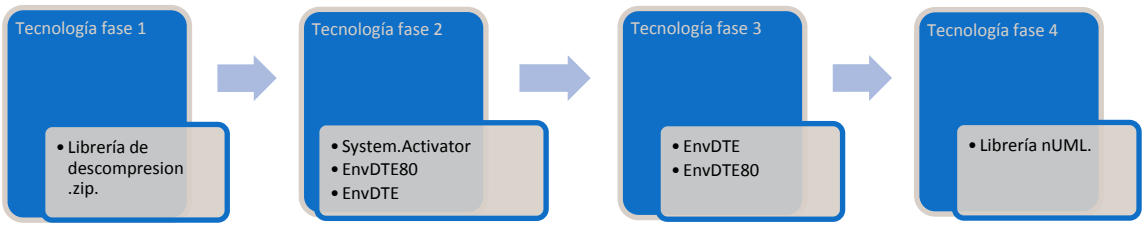

Ilustración 11: Tecnologías empleadas por fase

A continuación se explicará con más detalle cada una de estas fases y las bibliotecas usadas. Se analizará cual ha sido la solución de diseño adoptada y la relación entre todas ellas.

Antes de describir las etapas, se listarán los elementos de código que se analizarán.

ELEMENTOS DETECTADOS

En este punto se listara la información que se obtiene de las soluciones de Visual Studio 2008 escritos en los lenguajes C# y VB.

CLASES

Se Identifica dentro de cada fichero VB las clases definidas en él ya sean una o varias. De cada clase se recoge:

Atributos: Dentro de los atributos se recoge el nombre, tipo y visibilidad.

Nombre: Nombre de la clase.

Visibilidad: Visibilidad de la clase si han declarado alguna. Si no se declara visibilidad EnvDTE entiende que la visibilidad es privada

Si es abstracta o no

Métodos: Dentro de cada método se recoge el valor de retorno, nombre, visibilidad, parámetros, tipo (si es una función constructora o no), si está sobrecargado.

Dentro de cada clase además de los elementos que la forman se recoge una serie de relaciones que la clase pueda tener:

Generalizaciones: En el caso de que esté heredando de una o más clases.

Asociaciones: En el caso de que la clase tenga alguna asociación con alguna otra clase.

Agregaciones: En el caso de que la clase tenga una relación de agregación con alguna otra clase.

Implementaciones: En el caso de que la clase este realizando alguna implementación de una interface.

Dependencias: En el caso de que existan dependencias con otras clases.

INTERFACES

Se identifica dentro de cada fichero las interfaces que hay declaradas ya sean una o varias.

De cada interfaz se recoges:

Nombre: Nombre de la interfaz.

Visibilidad: Visibilidad de la interfaz.

Métodos: Los métodos que están definidos dentro de la interfaz. Las características asociadas a los métodos son las mismas que en el caso de las clases.

DESCOMPRESION DE FICHERO .ZIP

Requisito indispensable es descomprimir un archivo .ZIP para obtener las soluciones .NET que en él haya.

Como se comenta a lo largo de esta documentación, el componente debe aceptar el paso de archivos .ZIP y de soluciones .NET. Lo primero que hace el indexador será el validar mediante la extensión del archivo pasado a qué grupo pertenece. Este punto solo aplicara para aquellos archivos pasados que tengan una extensión .ZIP

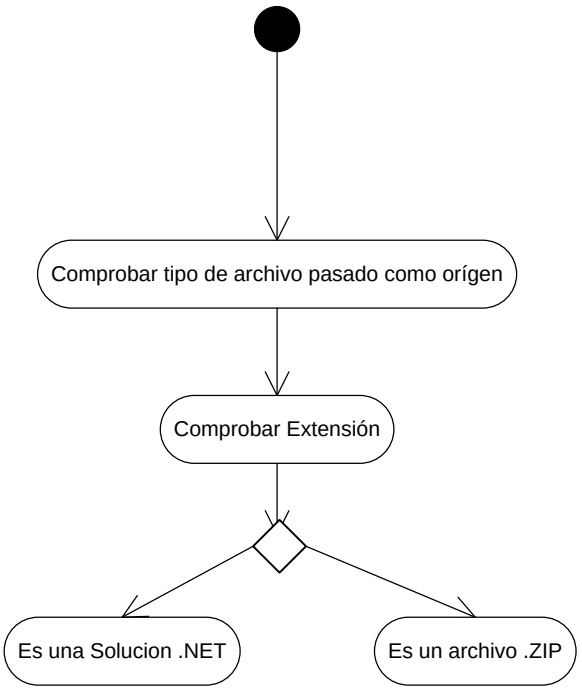

Ilustración 12: Comprobación de la entrada

Se desea depender lo menos posible de aplicaciones externas por lo que es recomendable no poner como requisito indispensable del entorno la instalación de una aplicación de descompresión de archivos. Por y para ello se busco a través de la red algún tipo de librería que nos facilitase ese trabajo y que simplemente referenciándola realizase la tarea solicitada.

El componente encontrado llamado Ionic.ZIP, es una librería libre que proporciona las rutinas necesarias para la descompresión de archivos .ZIP.

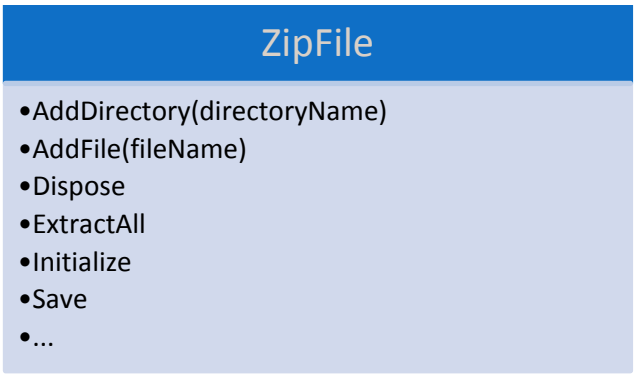

Ilustración 13: Componete ZipFile

A continuación se muestra la forma en la que el indexador, componente principal del proyecto descomprime los archivos:

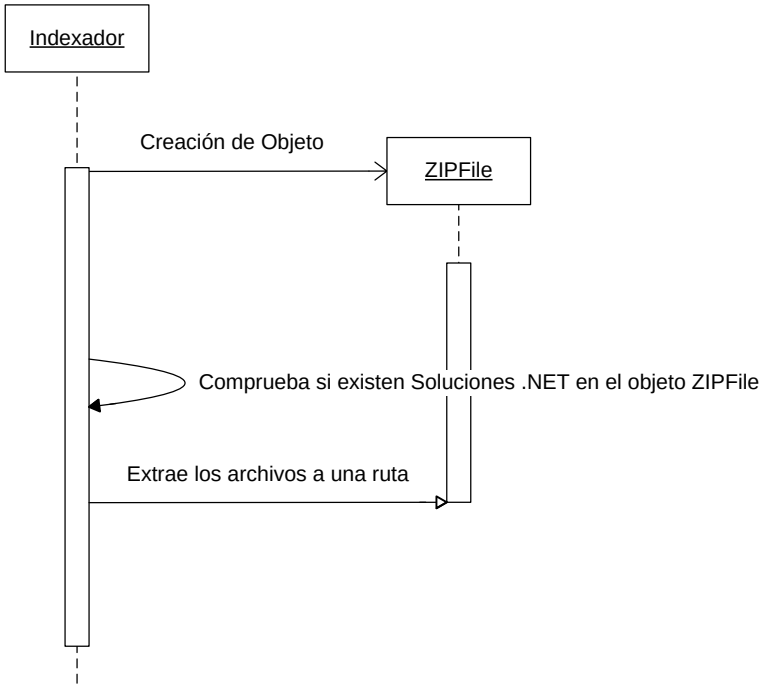

Ilustración 14: Diagrama de secuencia de la descompresión de archivos .zip

Este diagrama esta realizado para el caso en que el indexador encuentre soluciones .ZIP dentro del fichero comprimido en caso contrario, es decir el archivo .ZIP que se pasa no contiene soluciones .NET, el Indexador no extrae los ficheros del archivo .ZIP y acaba su ejecución.

LANZAMIENTO DE VISUAL STUDIO 2008 EN BACKGROUND DESDE EL CODIGO

Una vez descomprimido el fichero .zip se puede empezar a manipular los ficheros .sln asociados a la solución. Para poder acceder a toda la información que la solución contiene es necesario crear una instancia de Visual Studio y abrirla en ella. Una vez abierta la solución en la instancia de Visual Studio se puede comenzar a indexación

CREACIÓN INSTANCIA VISUAL STUDIO

Para poder crea una instancia de Visual Studio en tiempo de ejecución se ha utilizado la clase **Sytem.Activator**. Esta clase permite crear tipos de objetos localmente y obtener una referencia al mismo para poder manipularlo.

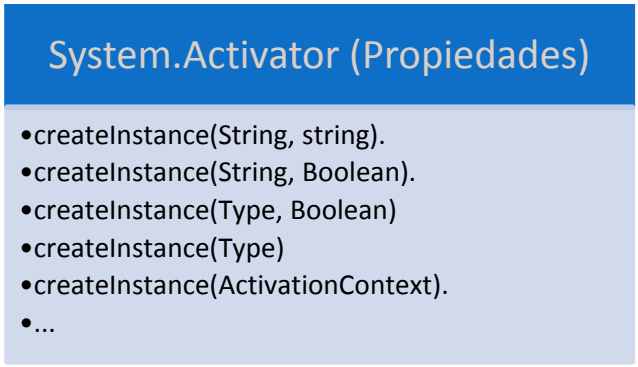

Ilustración 15: Componente Activator

De entre todos los métodos que la clase ofrece se han utilizado createInstance (Type), donde type indica el tipo de objeto que se quiere crear, que en este caso es “Visual.Studio.DTE.9.0”. El método devolverá una referencia al objeto creado para que podamos manipularlo. Dado que se ha especificado que el tipo de objeto que se quiere crear es Visual.Studio.DTE.9.0 se obtendrá una referencia a un objeto de tipo EnvDTE80.DTE2 que es el objeto de nivel superior del modelo de objetos de automatización de Visual Studio.

De esta manera se tendrá una instancia de Visual Studio sobre la que abrir la solución y además acceder a esa instancia para configurarla según criterio. Por último solo queda modificar dos propiedades del objeto DTE2 para asegurar que la instancia que se ha creado permanezca en modo oculto. Estas propiedades son:

- SuppressUI: A la que se asigna el valor true.
- UserControl: A la que se asigna el valor false.

Gracias a esto se tiene una instancia de Visual Studio 2008 ejecutándose en la máquina pero de manera totalmente transparente al usuario.

Para asegurar que no quedan procesos zombis en la máquina una vez que ya no nos sea necesario el objeto DTE2 se utiliza el método “Quit” para liberar los recursos usados.

APERTURA DE LA SOLUCIÓN EN LA INSTANCIA DE VISUAL STUDIO CREADA.

El siguiente paso antes de comenzar la indexación será abrir la solución en la instancia de Visual Studio que se acaba de crear. La clase EnvDTE.Solution permite crear un objeto de tipo solución asociado al objeto de Visual Studio, de esta forma una vez creado un objeto de tipo Solution se puede utilizar el método “Open” , pasándole como parámetro la ruta del fichero .sln para abrir finalmente la solución.

Queda un paso más para poder tener acceso total a la solución y a todos sus recursos. Hasta ahora se ha conseguido cargar la solución y todos los proyectos que contenga en la instancia de Visual Studio, sin embargo se sigue sin tener acceso a los ficheros de código que esta solución posee contenidos dentro de los proyectos. Sin este acceso no se podrá realizar la indexación ya que son los ficheros con extensión “.vb” los que contienen toda la información a la que se quiere tener acceso. Para poder analizar el código de estos ficheros se debe abrir de forma explícita esto es, se debe acceder uno por uno a todos los elementos del proyecto y en el caso de que sea un fichero de código utilizar el comando “Open” de la clase EnvDTE.ProjectItem para cargarlo.

A partir de aquí, una vez que todos los ficheros de código encontrados en la solución han sido abiertos, se está preparado para seguir con la siguiente fase, la indexación de toda la información que la solución contiene.

DIAGRAMA DE SECUENCIA DE LA CARGA DE LA SOLUCIÓN EN VISUAL STUDIO

El diagrama de secuencia para esta fase sería el siguiente.

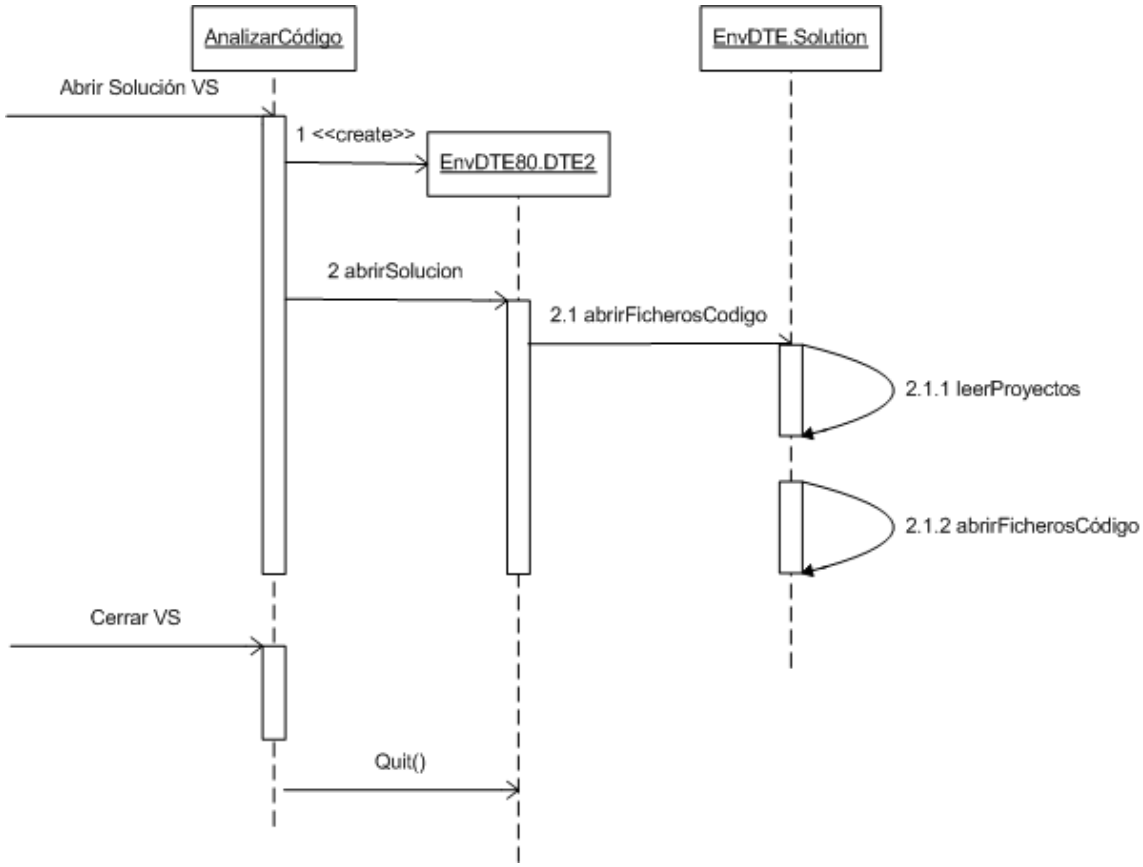

Ilustración 16: Diagrama de secuencia de carga de solución en Visual Studio

USO DE COMPONENTE ENVDTE/ENVDTE80

Una vez abierta la solución en Visual Studio, y comprobado que todos los ficheros de código han sido cargados correctamente se pasa a obtener la información que contiene. Para ello es necesario que de entre todos los ficheros que han sido cargados con la solución se seleccione para indexar solo aquellos que nos aportarán valor semántico, es decir los ficheros de código.

El método *AnalizarCódigo*, será el encargado de comenzar el proceso de recopilación de información recibiendo como parámetro la referencia a la instancia de Visual Studio. Como se indicó anteriormente la instancia de Visual Studio se representa gracias a la interfaz DTE2.

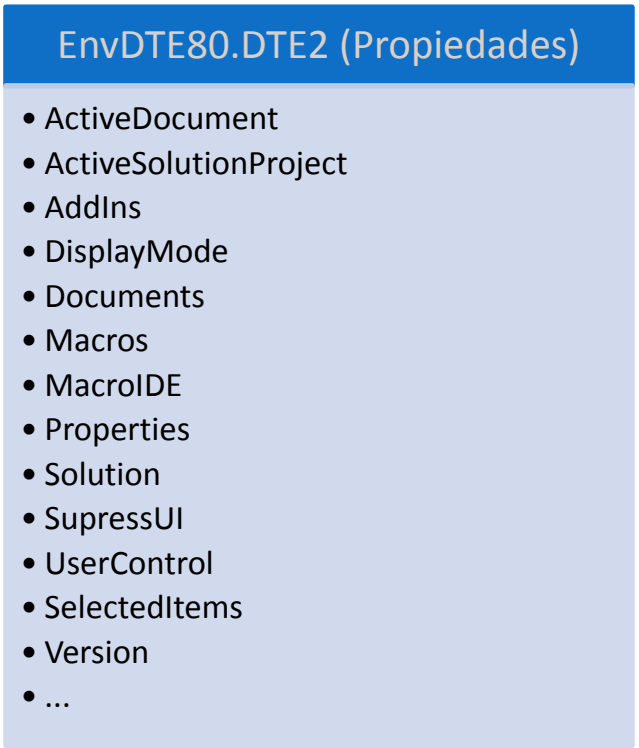

Ilustración 17: Componente DTE2

De entre todas las propiedades a las que se tiene acceso desde la interfaz DTE2 cabe destacar las siguientes:

- **ActiveDocument**: Obtiene el documento activo.
- **ActiveSolutionProjects**: Obtiene una matriz de proyectos seleccionados actualmente.
- **AddIns**: Obtiene la colección de addIns, que contiene todos los complementos disponibles.
- **Documents:** Obtiene la colección de documentos abiertos en el entorno de desarrollo
- **Solution**: Obtiene el objeto Solution que representa todos los proyectos abiertos en la instancia actual del entorno y permite el acceso a los objetos de generación.
- **SupressUI**: Obtiene un valor que indica si se debe mostrar la interfaz de usuario.
- **UserControl**: Obtiene un valor que indica si el entorno fue iniciado por un usuario o por automatización.
- **Version**: Obtiene el número de versión de la aplicación host.

Gracias a la propiedad Documents se obtendrá la colección de todos los documentos abiertos en nuestra instancia de Visual Studio, de esta forma se pasa de manejar la instancia de Visual Studio a una colección de Documentos abiertos en la solución.

De la clase Document ya se puede comenzar a extraer información útil y que nos aporta valor semántico. La siguiente figura muestra las propiedades más importantes de esta clase.

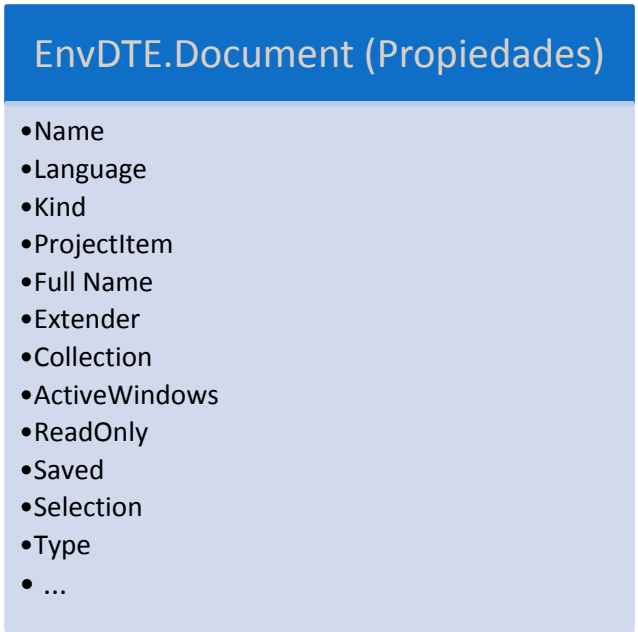

Ilustración 18: Componente EnvDTE.Document

De entre estas propiedades se destacan:

- **Name**: Obtiene el nombre del documento.
- **Kind**: Obtiene el tipo de documento.
- **Language**: Obtiene el lenguaje del documento. En nuestro caso nos permitirá saber si el documento está escrito en C# o VB por ejemplo.
- **ProjectItem**: Obtiene los objetos de proyectos asociados a nuestro documento.

Todavía es necesario profundizar más para poder tener acceso a toda la información contenida en el fichero de código y esto se conseguirá accediendo a la propiedad ProjectItem que devuelve todos los objetos asociados al documento.

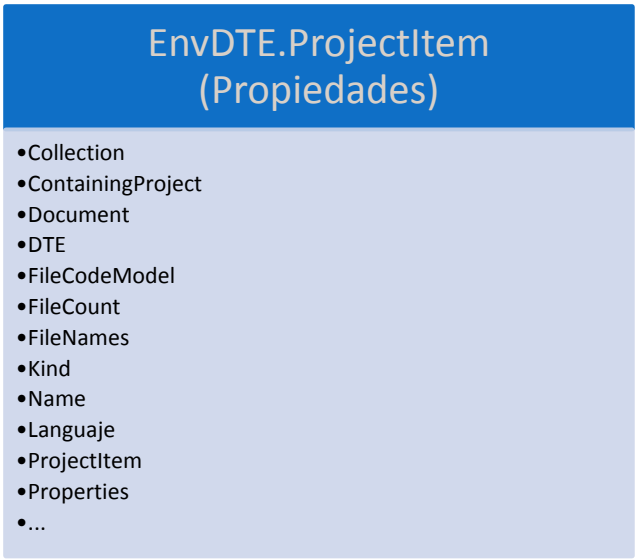

Ilustración 19: Componente EnvDTE.ProyectItem

De entre todas las propiedades de esta clase solo es relevante una: FileCodeModel. Esta clase representa el código contenido dentro de un documento de código y es por lo tanto esta clase, la que contiene toda la información que la aplicación debe indexar. Toda esta información se encuentra recogida dentro de la clase FileCodeModel en su propiedad CodeElements que devuelve una colección con todos los elementos de código.

La siguiente figura muestra el proceso seguido desde la instancia de Visual Studio hasta poder acceder a los elementos de código.

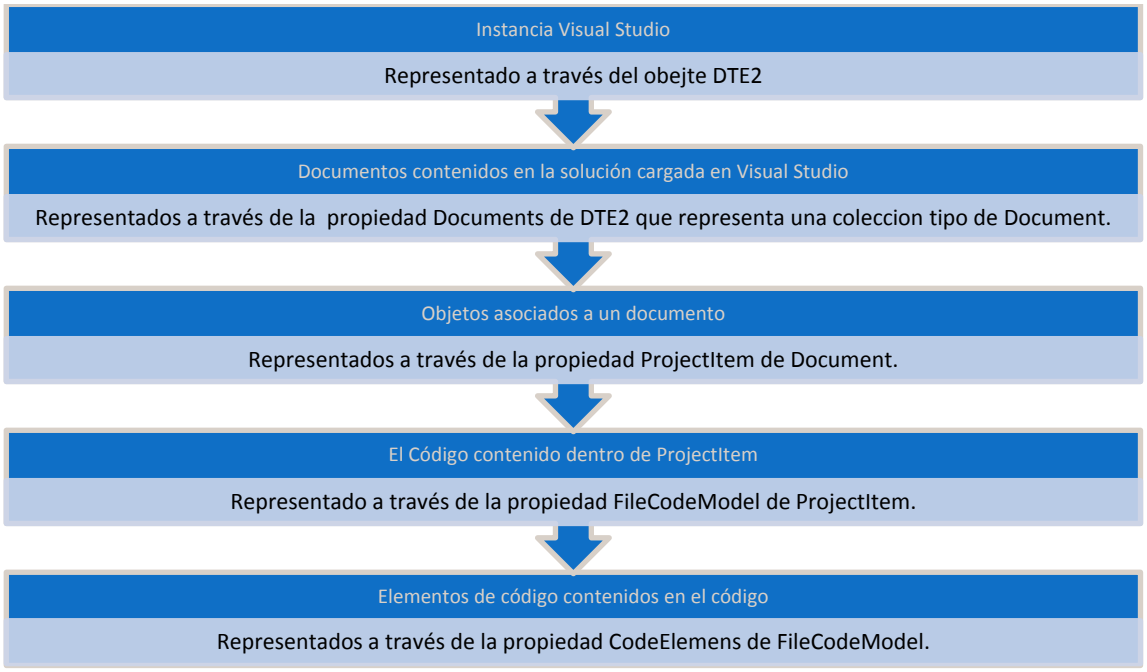

Ilustración 20: Esquema de acceso a los elementos de código

De esta forma a partir de los CodeElements se puede comenzar el proceso de indexación del documento. Este proceso consistirá en la identificación de los CodeElements que representan la información que la aplicación debe ser capaz de reconocer y que ha sido especificada en la especificación de requisitos

INDEXACIÓN DE LA INFORMACIÓN DE LOS FICHEROS DE CÓDIGO

Una vez llegado hasta los CodeElement, los mínimos elementos de información que se puede analizar, se puede comenzar con el proceso de indexación. El hecho de que se esté trabajando ya con elementos de un nivel tan bajo de abstracción permite identificar, en los ficheros de código, aquellos elementos de información que fueron especificados en los requisitos de software. Este es un paso crítico para el funcionamiento del sistema ya que del correcto análisis e identificación de elementos y relaciones presentes en el código dependerá la generación del metamodelo nUML. Si no se consigue generar un

pre-modelo de información completo, el metamodelo nUML no recogerá toda la información semántica que los ficheros de código ofrecen.

Esencialmente este proceso de análisis en el que se fundamenta la fase de indexación está basado en:

- **Identificación de los elementos de código:** Elementos que componen por sí solos el código y que son identificables en primera instancia, tal como: clases, atributo o métodos.
- **Identificación de relaciones existentes en el código:** Relaciones que se dan entre los elementos de código identificados tal como: dependencias, generalizaciones o asociaciones.

IDENTIFICACIÓN DE LOS ELEMENTOS DE CÓDIGO

Los elementos de código son directamente identificables y están relacionados uno a uno con cada CodeElement, es decir cada CodeElement representa un elemento de código. De esta forma el proceso de identificación consiste en comprobar que la clase de un CodeElement corresponde con alguno de los elementos de información que la aplicación tiene que ser capaz de identificar.

Para poder realizar este filtrado entre los CodeElement y recoger únicamente la información de aquellos que son útiles usamos la propiedad Kind. La siguiente tabla muestra la relación entre la propiedad Kind y los elementos de información que el sistema debe identificar.

Tabla 2: Equivalencia entre unidades de información

KIND	UNIDADES DE INFORMACIÓN
`vsCMElement.vsCMElementNamespace`	Namespace
`vsCMElement.vsCMElementVariable`	Atributo de Clase
`vsCMElement.vsCMElementClass`	Clase

vsCMElement.vsCMElementInterface	Interfaz
vsCMElement.vsCMElementFunction	Método

Estos son los cinco tipos principales de unidades de información que se identifican en los CodeElement. Profundizando en ellos se puede reconocer más elementos de información que nuestro sistema debe representar. La siguiente tabla muestra de manera ampliada la unidad primaria de información asociado con los elementos de información que se pueden obtener de ellos.

Tabla 3: Componentes de las fragmentos de código

KIND	UNIDADES INFORMACION	ELEMENTOS INFORMACION ASOCIADOS
vsCMElement.vsCMElementVariable	Atributo de Clase	Nombre Tipo Visibilidad Si es tipo primitivo Comentarios
vsCMElement.vsCMElementClass	Clase	Nombre Si es Abstracta Visibilidad

		Comentarios
`vsCMElement.vsCMElementInterface`	Interfaz	Nombre Visibilidad Comentarios
`vsCMElement.vsCMElementFunction`	Método	Nombre Visibilidad Parámetro salida Parámetros entrada

IDENTIFICACIÓN DE LAS RELACIONES EXISTENTES EN EL CÓDIGO

Una vez identificados los elementos de código de los que están compuestos los ficheros .vb de la solución se debe establecer las relaciones existentes entre ellos que también aportan un valor semántico muy importante al resultado del análisis.

Así pues asociado a cada elemento de código se pueden extraer las siguientes relaciones:

Tabla 4: Relaciones por elemento de código

ELEMENTO DE CODIGO	RELACION IDENTIFICABLE
Atributo	Asociación.
Clase	Generalización, Interfaces implementados.
Método	Dependencias.

NUML

La utilización de nUML es una parte fundamental de este proyecto. El objetivo prioritario de la librería es el de devolver la información mediante este componente.

Para describir y explicar cómo se ha utilizado este componente se va a dividir este punto en dos más pequeños.

En el primero se listara la librería nUML y los espacios de nombres que la forman.

En el segundo se explicara cómo se ha utilizado dicha librería en este proyecto

COMPONENTES DE NUML

El uso de nUML se basa en distintos espacios de nombres separados por su utilidad, lo mejor para su comprensión es ir explicándola una a una:

NUML.UML2

Este espacio de nombres contiene las clases básicas que componen UML 2.0. A continuación se presentan algunas de clases que este componente contiene:

- **Class**
- **Operation**
- **Action**
- **Activity**
- **Actor**
- **Relationship**
- **Comment**
- **Collaboration**
- **Call**
- **Model**

NUML.NUMLOPERATION

Este espacio de nombre contiene todas las operaciones necesarias para la creación de los distintos objetos de nUML. Dentro de este componente la clase que va a ser más utilizado en este proyecto será nUML.nUMLOperation.ModelManagement ya que esta clase contiene todas las operaciones estáticas que permiten indexar la información de las soluciones.

A continuación se pasa a listar algunas de las operaciones que se incluyen dentro de la clase:

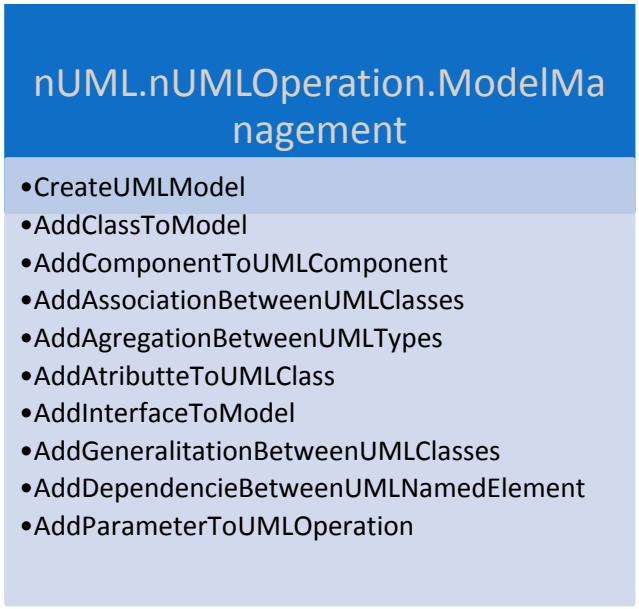

Ilustración 21: Componente nUMLOperation.ModelManagement

USO DE LA LIBRERÍA NUML

Ahora se expondrá ahora la forma en la que se utiliza la librería nUML para reflejar los elementos de código que anteriormente se han enumerado, en el modelo nUML correspondiente.

Como punto de partida indicar que el componente utiliza la interfaz ***Transformer*** por ello implementa los siguientes métodos:

GetTrasnformLevel: Indica si el archivo pasado por parámetro puede ser analizado o no.

ReleaseApplications: Se liberan todo los recursos utilizados por el componente.

TrasnformalInternal: Método principal, es el encargado de analizar, obtener y devolver toda la información de las soluciones pasadas por parámetros. El modelo obtenido es en nUML.

Se identificará a continuación una a una las operaciones nUML que se utilizan para la obtención de los modelos nUML.

Para crear el modelo se utiliza la operación: ***createUMLModel (nombre)***.

Nombre: Como nombre de modelo se introduce el nombre de la solución ya que como se comento anteriormente se ha equiparado el concepto modelo con el de solución.

CLASES

Para añadir una clase se utiliza: ***addClassToUMLModel (nombre, modelo)***.

Nombre: Nombre de la clase.

Modelo: El modelo nUML al que se va a insertar la clase.

Dentro de la clase se definirán las siguientes propiedades.

Visibility: Es decir el grado de visibilidad de la clase,

IsAbstract: Si la clase es abstracta

ATRIBUTOS

Para definir un atributo se emplea: ***addAttributetoUMLClass (nombreAtrib, clase)***.

NombreAtrib: El nombre del atributo.

Clase: Clase nUML en la que se define el atributo.

Para cada atributo se define su atributo **visibility**.

METODOS

Para incluir un método dentro de una clase se utiliza la función ***addOperation to nUMLClass (nombremetodo, parametrosmetodo, clasedeparametro, tipo de parámetro, clasenUML)*.**

NombreMetodo: El nombre del método.

Parámetros: Array con el nombre de los parámetros del método.

ClaseDeParametro: Si el parámetro es de entrada o saliida.

TipoParametro: Array con los tipos de los parámetros.

Una vez definidos los elementos estructurales de la clase definimos las relaciones que esa clase tiene.

DEPENDENCIAS

Dependencias: usamos **addDependecyBetweenUMLNamedElement (modelo, elemento1, elemento2)**.

Modelo: modelo nUML.

Elemento1: Clase que en ese momento se está analizando.

Elemento2: Clase con la que tiene dependencia. Si la clase todavía no existía en el modelo se inserta.

ASOCIACIONES

Asociación: se usa **addAssociationBeteweenUMLClases (clase1, rolclse1, clase2, rolClase2, modelo, nombreAsociacion)**.

Clase1: La clase que actualmente se esta analizando.

rolClase1: Nombre de la clase.

Clase2: La clase con la que se hace la asociación.

rolClase2: Nombre de la clase.

Modelo: el modelo nUML.

NombreAsociacion: Se está pasando actualmente “de atributo”.

AGREGACIONES

Agregaciones: se usa **addAgreggationBetweenUMLTypes (clasenUML1, rol1, claseUML2, rol2, modelo, nombre)**.

clasenUML1: La clase que actualmente se esta analizando.

rol1: Nombre de la clase que actualmente se esta utilizando.

claseUML2: La clase con la que se hace la agregación.

Rol2: Rol de la clase con la que se tiene la reliación.

Modelo: el modelo nUML.

Nombre: Se pasa de atributo.

GENERALIZACIONES

Generalizaciones: se usa **addGeneralizationBetweenUMLClases (modelo, clasepadre, subclase)**.

Modelo: modelo nUML.

ClasePadre: La clase de la que hereda

Subclase: La clase que se esta analizando.

REALIZACIONES

Implements: para definir un interfaz implementado por la clase se usa **addInterfaceRealizationtoClass (clase, interfaz).**

Clase: Clase nUML.

Interfaz: Interfaz nUML.

INTERFACES

Para definir las interfaces se usa **addInterfacetoUMLModel (nombre, modelo).**

Nombre: nombre de la interfaz.

Modelo: modelo nUML.

Se define la **visibility** de la interfaz.

Para definir los métodos de la interfaz se usa **addOperationToUMLInterface**.

El funcionamiento básico para estas operaciones viene definido según este diagrama:

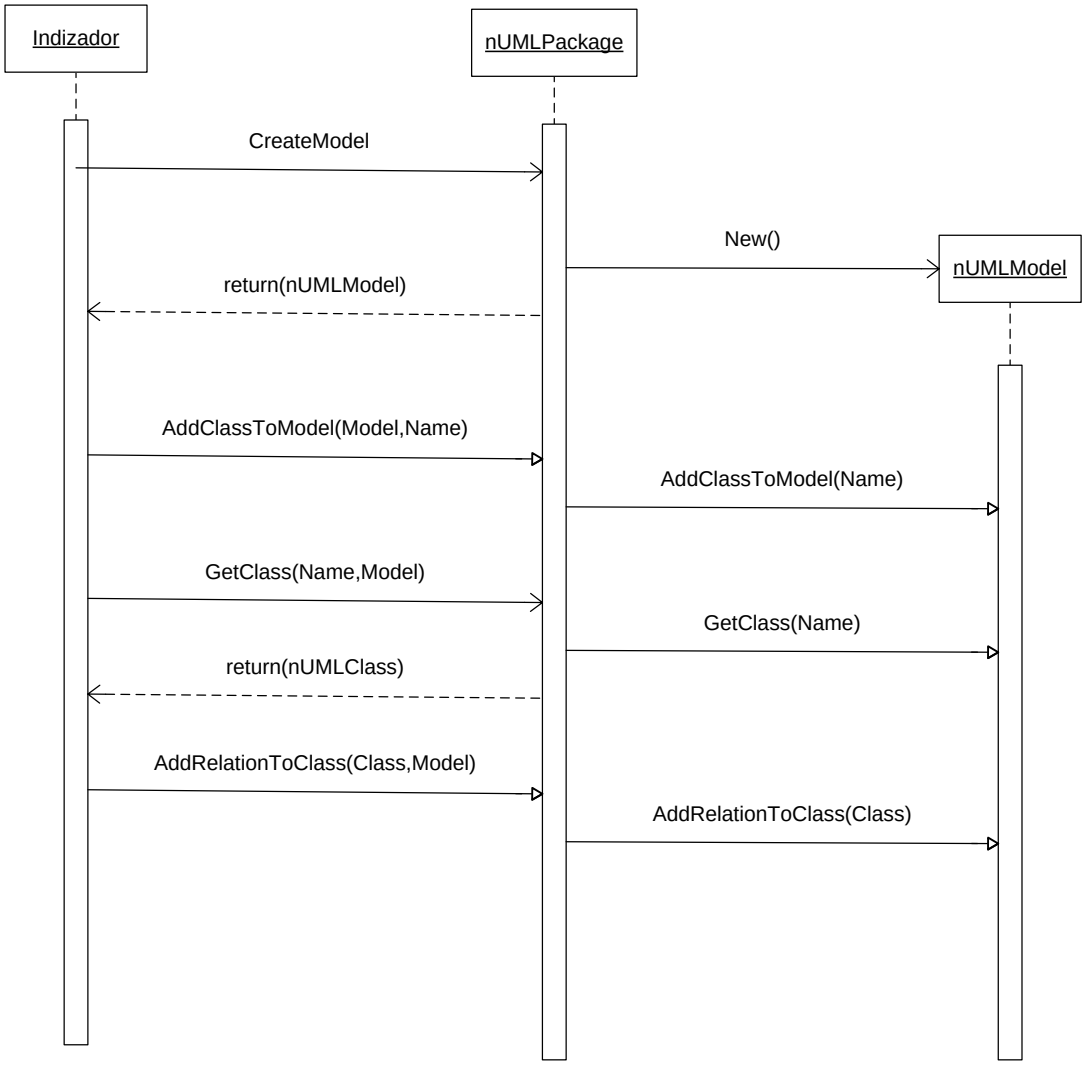

Ilustración 22: diagrama de secuencia de inserción de elemento en nUML

A continuación se muestra el diagrama e clases de la DLL resultante:

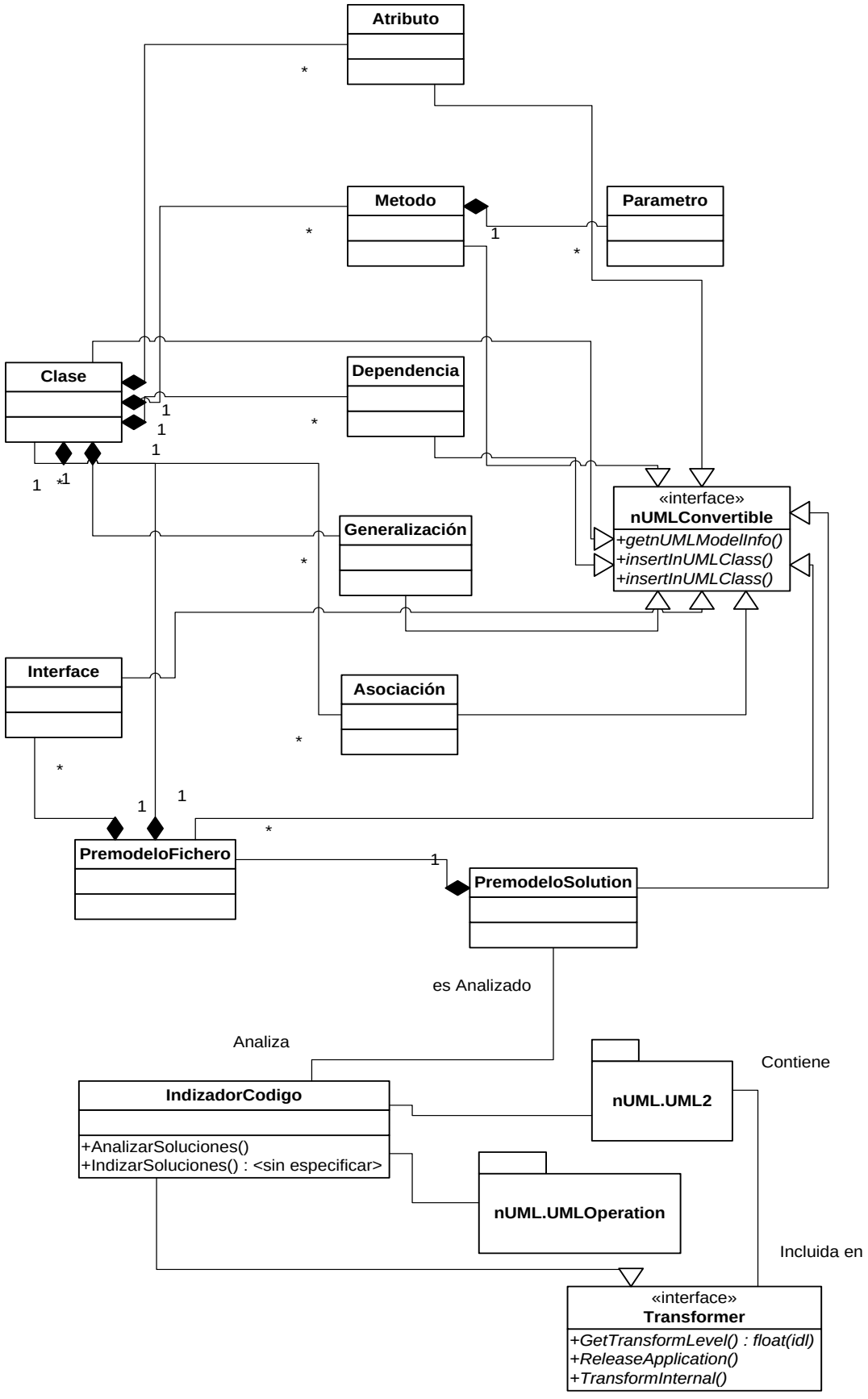

Ilustración 23: Diagrama de clases

GESTION DE COSTES

DISTRIBUCIÓN DE PLAZOS

La construcción del indizador de soluciones solicitado comenzó el día 15 de Abril del 2010, y finaliza el 25 de Septiembre del 2010.

Debido a que los desarrolladores de este proyecto compatibilizan este desarrollo con una vida laboral, se interrumpió aproximadamente 15 días en Agosto. Además de ello, debido a que toda la comunicación con el cliente se realizo por medio de email, el delay en puntos clave provoco un retraso de otro mes por causas irrelevantes para esta memoria.

Por lo tanto el esfuerzo empleado ha sido:

- De 15 de Abril de 2010 a 10 de Agosto de 2010, y del 23 de Agosto de 2010 al 20 de Septiembre del 2010.
- Jornada laboral de 4 horas.
- 5 días laborales a la semana, festivos incluidos.

Dentro de este control se incluye el desarrollo de la documentación necesaria.

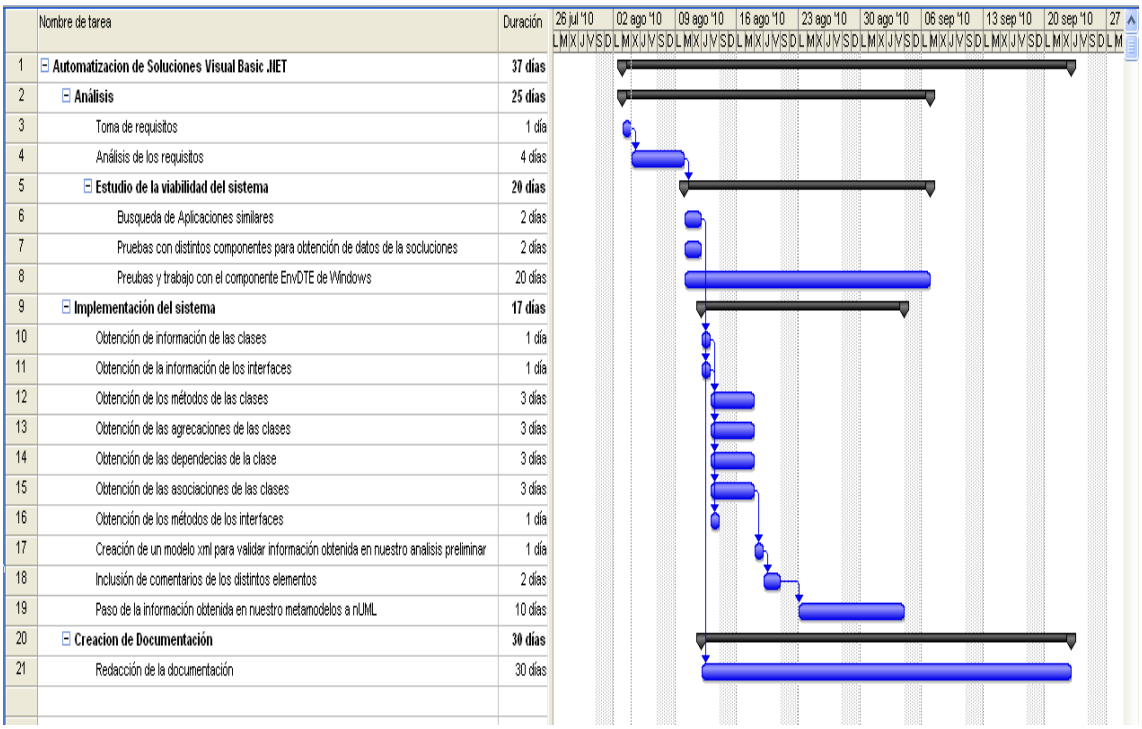

Ilustración 24: Diagrama de Gantt

El personal involucrado ha sido:

- Dos analistas programadores.
- Un director de proyecto.

A continuación se ofrece el resumen de todas las horas invertidas en el presente proyecto:

Tabla 5: Esfuerzo empleado

Fase	Nº total de días	Nº total de horas
Desarrollo del sistema	74	296
Redacción de la memoria	25	100
Reuniones del equipo de desarrollo	10	40

PRESUPUESTO

Es necesario incluir además el costo de las horas y de los materiales utilizados para el desarrollo del proyecto.

RECURSOS HUMANOS

Dentro del desarrollo se encuentra personal con distintas características y conocimientos y debido a ello con distintos honorarios lo que hace necesario la inclusión de las tarifas de los mismos.

Tabla 6: Tabla de salarios

Perfil en informática	Características	Salario Bruto Anual
Analista Programador	Experiencia entre 2 y 5 años	24000
Director de Proyecto	Experiencia de más de 10 años	40000

Teniendo en cuenta que el número de horas por convenio son aproximadamente 1810 para el sector de Técnicos el coste hora quedaría:

Tabla 7: Coste de recurso por hora

Perfil en informática	Características	Coste Hora
Analista Programador	Experiencia entre 2 y 5 años	14
Director de Proyecto	Experiencia de más de 10 años	22

Según la implicación que ha tenido cada uno y a partir de la información recogida en el punto anterior, es posible calcular el coste total en recursos humanos.

Tabla 8: Coste total humano

Miembro del equipo	Nº de horas	Coste total (€)
Analista Programador 1	200	2800
Analista Programador 2	200	2800
Director de Proyecto	36	796
	TOTAL (sin IVA)	6392
	TOTAL (con IVA)	7542,50

RECURSOS HARDWARE

A continuación se detalla la inversión realizada en los componentes hardware utilizados durante todo el proceso de desarrollo.

La vida útil de un portátil se puede estimar en 2 años, al tener un año un tortal de 1750 horas al contemplar el proyecto de 436 multiplicaremos el valor por ese ratio (0,125)

Tabla 9: Recursos Hardware

Componente	Coste Total(€)	Coste de amortización
Ordenador portátil	1000,00	125,00
TOTAL (con IVA)	1000,00	125,00

RECURSOS SOFTWARE

Para este punto se tienen en cuenta las licencias de las aplicaciones empleadas en el proceso de desarrollo del componente.

Aplicamos el mismo porcentaje que con el ordenador

Tabla 10: Recursos Software

Recurso	Coste (€/licencia)	Coste a.(€)
Sistema operativo Windows XP Professional	194 €	24,25 €
Entorno de desarrollo Microsoft Visual Studio .NET 2008	667€	83,40 €
Microsoft Office 2007	314€	39,25 €

Microsoft Visio 2007		667€	83,40 €
Microsoft Project 2007		313€	39,25 €
	TOTAL (con IVA)	2155€	269,55 €

FUNGIBLES

Tabla 11: Fungibles

Recurso	Unidades	Coste/unidad	Coste/recurso
Paquete de folios (500 folios)	1	6€	6€
Almacenamiento: Pendrive	1	25€	25€
Cuaderno hojas cuadriculadas	2	2€	4€
Bolígrafos	5	0,50€	2,5€
Impresión Documentación	4	12€	48€
	TOTAL (con IVA)		85,5€

FORMACIÓN

Debido a la naturaleza del proyecto, se considera que el coste de la formación de los miembros del equipo de desarrollo queda incluido en los anteriores apartados.

RESUMEN DE COSTES

Tabla 12: Resumen de costes

Concepto	**Coste**
Recursos humanos	7542,50
Hardware	125
Software	269,55 €
Fungibles	85,5
TOTAL (con IVA)	8022,55

RESULTADOS, CONCLUSIONES Y FUTUROS DESARROLLOS

RESULTADOS

Para finalizar el proyecto se ha creado una solución que cumple con todos los requisitos solicitados dando como resultado una DLL que es capaz de procesar una solución .net extrayendo de esta todo el contenido semántico que puede resultar reutilizable a futuro.

Esta solución se integra en un proyecto mayor que incluye interfaces que pintan el UML, de esta forma la DLL generada contribuirá a desarrolladores a generar sus diagramas una vez que hayan implementado las soluciones. También servirá para almacenar de una manera fácilmente reutilizable los metamodelos de las soluciones .net de las que se dispongan.

La DLL resultante es capaz de descomprimir ficheros y analizar que estos contengan soluciones .net e indizar dichas soluciones.

CONCLUSIONES

Tras el desarrollo de este proyecto y el conocimiento adquirido sobre reutilización de código y sobre la aplicación de reutilización en la que se encuadra este proyecto se llega a las siguientes conclusiones:

- Gran parte del tiempo empleado en la codificación de código por parte de los creadores se emplea en implementar funciones similares que ya ha realizado.
- El almacenamiento de la metainformacion de los distintos segmentos de código es más útil, mas intuitivo y ocupa menos que almacenar todo el código, además de ser mas fácil de indizar conceptualmente.
- La utilización, estandarización e inclusión del concepto de reutilización dentro del seno de las compañías de desarrollo software provocaría una mejora dentro de los procesos de codificación, además de ganar en velocidad y por lo tanto economizar coste, solo existiría la contrapuesta del costo del software de reutilización y la curva de aprendizaje de ellas, costo de estos dos últimos que se diluiría en un tiempo muy contenido debido a las actuales tarifas de costo de las horas de los recursos.
- La reutilización de código se puede ampliar a la reutilización de estructuras y buenas formas de programación.
- La reutilización debe considerarse como un concepto global y no personal, es decir, es tan importante el reutilizar los conceptos adquiridos dentro del mundo heterogéneo de la informática como la reutilización de nuestro código desarrollado.

FUTUROS DESARROLLOS

Debido a que el componente actual está encuadrado dentro de un proyecto más grande, se van a incluir en este punto los posibles desarrollos de esta.

En este componente se ha incluido la funcionalidad de descompresión de soluciones .ZIP, dicha funcionalidad podía ser asumida como un paso anterior a la obtención de información por la aplicación superior de tal manera que, ya que es conocido que existen distintos indizadores para otros lenguajes, se puedan obtener de archivos .ZIP no solo las soluciones de .Net escritas en VB si no que se pueda indizar la información de por ejemplo soluciones de C#, XML o cualquier otro lenguaje del que se disponga de un indizador.

Debido a la arquitectura de la aplicación superior se pueden desarrollar tantos indizadores distintos como lenguajes de programación hay en el mundo.

Recordar por otro lado que UML no solo es un lenguaje de especificación aplicaciones informáticas si no que es capaz de describir prácticamente cualquier sistema, por ello se puede intentar abordar indizadores de los distintos formatos de especificación de sistemas, ya sea una analizador de texto plano capaz de obtener palabras claves o un indizador capaz de obtener información de otros formatos gráficos de definición de sistemas distintos a UML.

Otra forma de ampliar la aplicación puede ser ampliando el espectro de formatos de compresión de manera que se pudiesen analizar más archivos de la red.

Un paso más seria el conseguir obtener aun más información del código de manera que se pudiesen crear otros diagramas que ayuden aun mas definir la solución.

BIBLIOGRAFIA

- http://salarios.infojobs.net/
- http://convenios.juridicas.com/convenios/ingenieria-y-oficinas-de-estudios-tecnicos-convenio-colectivo-2007-2009-espana.html#a23
- http://msdn.microsoft.com
- http://www.uml.org/
- http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado
- http://office.microsoft.com/es-es/visio-help/CH010064890.aspx
- http://www.codeproject.com/
- http://bauer.ie.inf.uc3m.es/
- http://convenios.juridicas.com/convenios-sectores.php
- http://modeling-languages.com/es/content/xmi2-una-herramienta-para-intercambiar-modelos-uml-entre-herramientas-case#try
- http://modeling-languages.com/es
- http://www.sparxsystems.es/New/products/ea.html

INDICES DE FIGURAS

INDICE DE TABLAS